FUNDAMENTAL DISTRIBUTED ALGORITHMS FOR MANAGEMENT OF COMPUTER NETWORKS WITH CHANGING TOPOLOGIES

BY
KENNETH CHIH-KEN LUO

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN
PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1991

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere gratitude to my dissertation advisor, Dr. Yuan-Chieh Chow, for his guidance and encouragement of this research, which was supported under the SDI/IST program, administered through U.S. Navy and U.S. Army. I also want to thank Dr. Richard Newman-Wolfe for numerous fruitful discussions which eventually led to some significant results of this research.

Other dissertation committee members also contributed to the work: Dr. Rick Smith made helpful suggestions on my research; I learned so much from his excellent course on analysis of algorithms. Dr. Yann-Hang Lee provided insight on some topics of the research. I am very grateful to both of them. Thanks are also due to Dr. Mark Yang of the statistics department for graciously consenting to be on my committee.

Finally, I would like to thank my wife, Sze-Mei, who has patiently watched this endeavor and given me the greatest encouragement.

TABLE OF CONTENTS

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

FUNDAMENTAL DISTRIBUTED ALGORITHMS FOR MANAGEMENT OF COMPUTER NETWORKS WITH CHANGING TOPOLOGIES

By

KENNETH CHIH-KEN LUO

December 1991

Chairman: Dr. Yuan-Chieh Chow
Major Department: Computer and Information Sciences

Distributed computing systems hosted in a network with changing topologies often encounter management problems which require the application of various control algorithms. Typical networks with such property include packet-radio networks, point-to-point satellite communication networks and networks with a high degree of link failures. In such networks, due to the instability of network topology, maintaining a reliable distributed computing environment is a complicated issue. There are a number of management problems inherent in these network systems. This research focuses on issues that are considered to be more fundamental.

In the first part of this research, we consider a network reconfiguration problem in network topology designs. The problem can be briefly described as follows: given a number of nodes in the network, each node has a fixed number of communication ports and is mobile. How can an algorithm be designed that reconfigures the network periodically such that the network always has maximum connectivity? We present a partitioning method to achieve this goal. The algorithm performs better than the best of the previous results in terms of time complexity.

The second issue addressed is a message broadcast problem. We consider a general model of a network with changing topologies, the "eventually connected" network model, i.e., networks that may be undergoing arbitrary topological changes but are not permanently disconnected. We focus on the issue of message broadcasts on such networks. A message broadcast protocol is a distributed algorithm that allows a message to be sent to every node in the network via the communication links connecting the nodes. It has been shown that in this kind of network, no conventional broadcast protocol is reliable. It has also been shown that a reliable broadcast protocol can be designed with unbounded message buffers. We improve this result by presenting an efficient reliable broadcast protocol using only bounded message buffers.

In the third part of this research, we consider a closely related problem of the message broadcast, the so-called global survey problem. That is, given a dynamic network which is subject to unpredictable link failures but remains connected and in which each node is unaware of the network topology, can a node broadcast a message to every node and then get a reply from every one? We provide a message-complexity lower bound as well as a time-complexity lower bound. An optimal protocol is also given.

In summary, this dissertation is devoted to a detailed and in-depth study of some fundamental issues involved in networks with changing topologies. Significant results have been produced that may have profound impacts on research areas such as network topology designs, network protocol designs, distributed databases and the theory of distributed computings.

CHAPTER I
INTRODUCTION

1.1 Scope and Objectives

A distributed system hosted in a network with changing topologies often encounters management problems which require the application of various control algorithms. Typical networks with such properties include packet-radio networks and point-to-point satellite communication networks which are among the main research areas of the Strategic Defense Initiative (SDI). In these networks, due to the instability of network topology, it is much more complicated to maintain a network system so as to provide a reliable distributed computing environment. There are a number of management problems inherent in these network systems. This research focus on several of them which are considered to be fundamental: network reconfiguration, message broadcast, and network surveys.

The solutions for these issues rely heavily on the design of effective and efficient distributed algorithms. Unlike those centralized algorithms that are executed as a single program by a single processor or by a number of processors but one at a time, a distributed algorithm is usually implemented as a node algorithm, i.e., each node in a network is installed with one copy of the algorithm. The algorithm works in an event-driven fashion; each node responds to a particular event by running a particular procedure. If no event occurs, the algorithm is idle.

An important feature of distributed algorithms is that the communication between nodes running the same algorithm is done by exchanging messages. Therefore, in addition to being correct and efficient, a distributed algorithm often needs to be designed in such a way that the total number of messages exchanged in a network is minimized, which further complicates the task of designing a good distributed algorithm. Some of the problems considered in this proposal require substantial modifications and extensions of the existing distributed algorithms which are usually intended for networks with fixed topologies; others are new problems for which no algorithm is available, let alone an efficient one. These problems are briefly discussed as follows.

1.2 Mobile Network Reconfiguration

A network with changing topologies needs to reconfigure its topology for the following two reasons: 1) Communication links may be broken due to the limited broadcasting power of each node, as shown in packet-radio networks, or due to the blocking of transmission by the Earth, as in the case of satellite networks in which a communication link between a pair of nodes is established by targeting laser transceivers toward each other. 2) The routing scheme of the network becomes inefficient. As topology keeps changing, so does the propagation delay of each link. Eventually, message routing becomes inefficient because previously computed routes are no longer optimal.

One of the most widely adopted criteria for measuring the performance of a reconfiguration algorithm is the connectivity of the network obtained. In this research, we present a new algorithm for this problem. The algorithm is based on network partitioning. Our results show that the algorithm yields optimal connectivity and is superior to existing algorithms in terms of time complexity [22].

1.3 Reliable Message Broadcast

For various reasons, a node in a network may need to broadcast messages to the other nodes. In networks with fixed topologies, messages are usually delivered through some broadcasting tree for the sake of reducing the number of messages exchanged. In unreliable networks, the task is fairly complicated.

Some sophisticated broadcast techniques have been developed to achieve reliable broadcast in networks with slowly changing topologies. This research focuses on developing a reliable and efficient broadcast protocol in networks where nodes are mobile with varying communication delay across each link, and where links may fail or recover in an unpredictable fashion. Links may fail or recover for two reasons: (1) due to an unplanned occurrence, i.e., a link breakdown, or (2) as part of a network reconfiguration scheme to update topology.

In this problem, a protocol is measured by three standard criteria: reliability, message delay, and communication cost. We present a reliable protocol which is simple, efficient, and robust. This protocol is a distributed algorithm which responds to the arrival of a message by performing special functions. Our protocol can achieve a minimum message delay with low communication cost [23]. In addition, we address the problem of message buffer utilization, which has never been successfully addressed in unreliable networks.

1.4 Global Network Survey

The network survey problem considered in the research can be briefly defined as follows. There exists a distinguished node in the network and it wants to glean some global information from other nodes in the network. The task of the node is to send a short message to every node in the network and get a response from each node in finite time. We call this a "network survey" which is conducted in an unreliable environment; in a network with changing topologies.

The network environment discussed in the topics is slightly different from the problem of network broadcast, in that the network, despite link failures, remains connected throughout the whole survey process. The significance of the problem is its close relationship with other fundamental problems such as network votings and distributed consensus.

Significant results have been produced on this problem. We have shown some lower bounds on the time and message complexities of this problem. Furthermore, a message-optimal algorithm has been developed during this research [24].

CHAPTER II
MOBILE NETWORK RECONFIGURATION

2.1 Background

The problem of network reconfiguration is considered a major issue in the backbone design (or topological design) of networks with rapidly changing topologies. For satellite networks connected through the use of laser transceivers, this problem is very critical for overall network performance. However, it is a complicated task to find an efficient and optimal algorithm for reconfiguration due to the following inherited properties: 1) variable propagation delay due to the changing lengths of the links, 2) visibility constraints due to the blocking of the Earth, i.e., not every pair of satellites can see each other, and 3) periodic link reconfiguration as the line of sight between two connected satellites may be blocked by the Earth over time. Since network reconfiguration in the laser-based point-to-point satellite networks involves link reassignment, we will use the two terms link assignment and reconfiguration interchangeably throughout this thesis.

A major issue in network reconfiguration is that of maximizing connectivity of a network. This problem has been studied extensively in point-to-point store-and-forward networks [6,19,21,34]. It can also be transformed into that of finding the maximum number of node-disjoint paths between every pair of nodes in a graph [13,21,31]. In particular, Schumacher [31] has given an $O(n^2)$ algorithm (where n is the number of nodes in the graph) for constructing k-regular graphs with maximum connectivity. The algorithm presented by Schumacher is the fastest algorithm as far as the time complexity is concerned. However, Schumatcher's approach is a pure graph algorithm in which every pair of nodes is connectable. Thus the algorithm is inapplicable (at least directly) to satellite network reconfigurations. The visibility constraint on the satellite computer networks makes it more difficult to reconfigure a network with respect to maximizing network connectivity because each node can see only a subset of the nodes. Relatively few works have been done on maximizing network connectivities for point-to-point satellite networks. McLochlin [25] and Ward [36] proposed methods to handle a special case of this link reconfiguration problem by imposing strict constraints on satellite orbits so that satellites circulate on several equally spaced planes and within each plane satellites are approximately equidistant from their neighbors.

Another issue in point-to-point satellite networks is to determine how and when reconfiguration is invoked. Research on other mobile networks also deals with the problem of network reconfiguring such as those for packet-radio networks [17,18,30]. Most of these works adopt the failure-driven reconfiguring approach, i.e., as long as the node or link is properly functioning, no reconfiguration is needed [7]. One criterion often adopted in packet-radio network reconfigurations is "minimum disturbance" [18] in the sense that reconfiguration normally requires updating or swapping links locally and thus the unaffected links should be preserved. In point-to-point satellite networks, this "minimum disturbance" is very difficult to obtain because most of the existing links will be blocked by the Earth sooner or later. Considering the periodic orbiting of satellites, the link reconfiguration needs to be carried out globally for there is no way to restrict the reconfiguration to a local level. Instead, we choose to do over-all network reconfiguration on a periodic basis to ensure that a network is highly connected all the time except at the moments of reconfiguration.

2.2 A Satellite Network Model

2.2.1 Assumptions

For systems as complex as satellite networks, it is necessary to make some basic assumptions so that solutions are feasible. Here we assume the following things hold:

(1) Global Knowledge – It is assumed that information about exact locations, directions of motion and other related data about the satellites can always be derived through mathematical computations based on orbital mechanics or alternatively from broadcasting of ground-based tracking stations.

(2) Fixed Number of Links at Each Satellite – Ward [36] has shown that it is most cost-effective if each satellite is installed with 4 transceivers. In our model we also assume that each satellite has exactly 4 transceivers. So, except in some unusual circumstances, the network can be regarded as a 4-regular graph, i.e., each node has exactly 4 edges.

(3) Global Time Scheme – Each satellite will need a clock synchronized to all the clocks on the other satellites. Since communication links between satellites are established by targeting laser transceivers to each other, some kind of synchronization is necessary. In this case we believe that using a global clock to drive the link assignment process is the simplest way to ensure reliable retargeting.

(4) Periodic Reconfiguration – The reconfiguration is done by retargeting transceivers among satellites to establish a new set of links on a periodic basis. Each satellite synchronizes its reconfiguring process to the others at regular periods. The reconfiguration is synchronized by having the satellites run their reconfiguring process simultaneously. During this short period of network reconfiguration, no packets can be exchanged between satellites. The advantage of periodic reconfiguration is that it allows satellites to perform more smoothly

by taking reconfiguration into account. For example, by knowing the next reconfiguration is approaching, a node may decide to disconnect the virtual circuits built on the present topology and/or set up some new ones based on the incoming topology.

2.2.2 Partitioning Space into Regions

A good solution to the link assignment problem should satisfy two goals – maximize connectivity and be computationally efficient. The approach we propose satisfies these requirements in handling large networks. In this method, a network is partitioned into small subnetworks. Each subnetwork independently generates a locally optimized topology in terms of connectivity. Then these subnetworks are connected together to form a complete network with maximum connectivity.

Applying this method to satellite networks, the space in which satellites orbit is partitioned into a number of regions of approximately the same size. Each region contains a subnetwork. For the sake of easy demonstration, the ensuing discussions assume that all satellites orbit at the same altitude. The multiple-altitude cases can be handled essentially in the same way. The size of a region is determined based on the principle that two satellites between a link are visible to each other all the time before the next reconfiguration occurs. The optimization of the region size will be discussed in detail in section 5.

To specify regions and satellites formally, a coordinate system (γ, θ, ψ), where γ represents the altitude, θ the latitude, and ψ the longitude, is adopted. Basically, the surface where satellites orbit is partitioned in such a way that the latitude is partitioned into twice as many intervals as is the longitude. For instance, if the longitude has x intervals, $[\psi_0, \psi_1]$, $[\psi_1, \psi_2]$, $[\psi_2, \psi_3]$, ... , $[\psi_{x-1}, \psi_x]$, then the latitude has 2 x intervals, $[\theta_0, \theta_1]$, $[\theta_1, \theta_2]$, $[\theta_2, \theta_3]$, ... , $[\theta_{2x-1}, \theta_{2x}]$. Note that, $\psi_0 = \psi_x$ and $\theta_0 = \theta_{2x}$ because the surface is round. So the network is partitioned into $2x^2$ regions.

Each region is represented by $([\theta_i, \theta_{i+1}], [\psi_j, \psi_{j+1}])$. Since we intend to make the area of each region approximately the same, these intervals are not of equal width. For example, the widths of θ-intervals near two poles, as their two ψ-borders merge into poles, should be greater than those of intervals in other areas.

Let $v_1, v_2, \ldots, v_n$ denote all satellites in the network, and $R_1, R_2, \ldots, R_m$ represent all regions partitioned from the spherical surface. A node v_i at $(\theta_i, \psi_i, \gamma_i)$ is in region R if $\theta_i \in [\theta_n(R_i), \theta_s(R_i)]$ and $\psi_i \in [\psi_e(R_i), \psi_w(R_i)]$ where $\theta_n(R_i)$, $\psi_e(R_i)$, $\theta_s(R_i)$ and $\psi_w(R_i)$ are the four borders of R_i. Ties are broken arbitrarily. Note that not every region has a northern or southern border. For instance, a region near a pole may lack a northern or southern border (in fact, the border reduces to a point), and the shape of a region in this area is much like a triangle on a spherical surface as illustrated in Figure 2.2.1. The shapes of the other regions are rectangular or trapezoidal, depending on their latitudes.

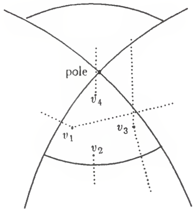

Figure 2.2.1 Regions and their connections near a pole.

2.2.3 Reconfiguring Networks

Since a global timing scheme is used, each satellite will be running the same link assignment algorithm simultaneously to reconfigure the network. The synchronization of link establishment for the network is realizable as long as each node keeps a copy of the whole network graph. When every node is executing the same algorithm, a consensus can be reached as each node knows which edges should be discarded and which edges should be set up.

The links formed by a satellite are determined by the region it is in and by its location within the region at the time of reconfiguration. Between reconfigurations (link assignments), it is permissible for a satellite to cross the borders into the other regions. It is assumed that transceivers on each link will keep pointing/tracking each other once the link is set up, regardless of where the other satellite is. If the region size and the reconfiguration period are properly determined, links will not be broken when satellites cross the borders.

2.2.4 Scheduling Network Reconfiguration

As has been pointed out, the reconfiguration of the network is done periodically. The next question is how often should a network be reconfigured. Since each reconfiguration requires a temporary halting of message exchanges, to maximize network utilization the interval between two consecutive reconfigurations should be as long as possible under the constraint that no links will be broken in between. Thus, the maximum reconfiguration period is the minimum of the longest time interval any two linked satellites can orbit and still maintain their common link. This can be determined after the region size has been decided. Figure 2.2.2 illustrates how the period is calculated. Suppose that the following regions R_1, R_2 each have only one node and at the time of a reconfiguration their positions are x_1 and y_1. Let x_2 and y_2 be two points in the space such that $\{x_2, y_2\} \in \{ \{x, y\} : | x - y | =$ the maximum distance that 2 satellites are visible to each other $\}$. Then the time for a satellite to travel from x_1 to x_2 or from y_1 to y_2 is the longest period of reconfiguration for which no links will be broken. This can be computed as soon as the region size and the orbits of the satellites are determined.

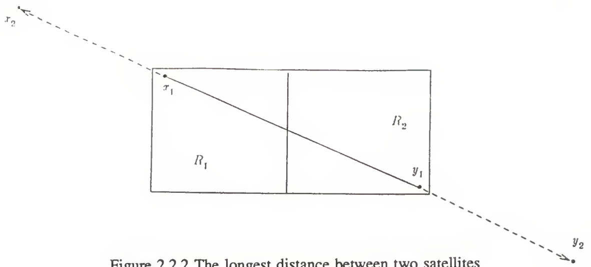

Figure 2.2.2 The longest distance between two satellites in adjacent regions

2.3. The Hierarchical Approach for Reconfiguration

Before we describe the hierarchical approach, let's introduce some terms. As the network is partitioned into regions, some of the nodes have to set up links across the regions so that packets can go through these links to other regions. Those nodes having border-crossing links are called ***backbone*** nodes and the others are called ***internal*** nodes or ***non-backbone*** nodes. Similarly, those border-crossing links are named as backbone links and the others are internal links. The role each node plays is not fixed. It depends on the location of the node. In general, each region will select nodes closest to the 4 borders (eastern, southern, western, and northern borders) to be its backbone nodes. Backbone nodes play more important roles in routing packets because inter-region packets will be forwarded via at least one of these nodes. In each region, except for the one-node case as shown in Fig 2.3.1.a, there are 6 backbone links; 4 links cross the northern and the southern borders; the other 2 links cross the eastern and the western borders. The distribution of links over borders is not unique. For instance, another alternative would be to put 4 links on the eastern and western borders and 2 links on the others. The basic principle ("single-hop" principle) is to allow the by-passing messages to be forwarded in one hop vertically as well as horizontally. The 6-backbone-links arrangement is derived partly from the 4-regular graph constraint and partly from empirical results of constructing primitive topologies for very small networks. There are possibly other arrangements with different numbers of backbone links in a region available to serve the purposes of obtaining regional maximum connectivity and of observing the single-hop principle as mentioned above.

The hierarchical link assignment algorithm can be divided into two levels. the first-level task is to determine a regional topology which includes establishing internal links and selecting backbone nodes. The second-level task is to combine regions into a connected network by setting up backbone links among backbone nodes of adjacent regions. The algorithm is designed to maximize regional connectivities as well as the whole network connectivity.

In the first-level task, regions are handled in two different ways: the primitive case and the recursive case. If a region contains 7 or fewer nodes, it is handled as the primitive case. Otherwise, it is regarded as the recursive case. The primitive case consists of 7 topologies. In the primitive case, the regional topology is predetermined and links are set up accordingly. In the recursive case, the link assignment algorithm works by "peeling off" layers of nodes, 4 nodes at a time, until it ends as one of 4 terminal cases; each layer contains the 4 nodes nearest to the 4 borders. The four terminal cases are shown in Figure 2.3.3.

2.3.1 Topologies for the Primitive Case

Seven primitive topologies are provided for the construction of regional subnetworks with fewer than 8 nodes. Figure 2.3.1 illustrates all of them. Note that these topologies are chosen to maximize the regional connectivity. By maximum regional connectivity, we actually mean that a node can construct 4 node-disjoint paths to any other node in the same region via the nodes in the same regions or in neighboring regions.

The primitive topologies are among the few topologies which meet the maximum connectivity requirement and the single-hop principle. The topologies have the following properties:

(1) no links are redundant in the sense that every transceiver is used and no more than one link is assigned to any pair of nodes;

(2) messages generated from a host inside a region can be forwarded efficiently out of this region via fewer than 3 intermediate nodes;

(3) "straight-line hops," east-west or north-south message deliveries, are favored. There are backbone nodes in each case (e.g., v_1 and v_2 in Figure 2.3.1.b, or v_1 and v_3 in Figure 2.3.1.c-g) that have 2 backbone links crossing east-west and north-south borders. Therefore, one-hop transmitting is allowed through this region. However, with the 4-link constraint, no similar "convenience" is offered if a by-passing message wants to "make a turn" through this region.

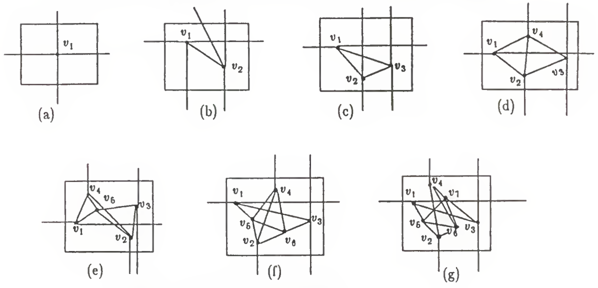

Figure 2.3.1 Primitive Topologies

2.3.2 Topologies for the Recursive Case

If the number of nodes in a region is greater than 7, the strategy is to construct a topology recursively. The link assignment algorithm works in the following manner: First, 4 nodes, each of them the nearest node to a border, are chosen to form the first layer of the region. Note that the nodes in the first layer will also be the backbone nodes of the region and will connect to backbone nodes in the neighboring regions. Second, the algorithm continues to peel off a set of 4-nearest-to-border nodes to form intermediate layers until one of the terminal cases is reached for which topologies exist. Figure 2.3.3 shows the topologies of all four permissible terminal cases. Finally, any two adjacent layers, i and i+1, are connected by their corresponding 4 nodes (e.g., $v_1^{(i)}$, $v_2^{(i)}$, ..., etc.) as shown in Figure 2.3.2.b where the $v_j^{(i)}$'s are of one layer above the $v_j^{(i+1)}$'s.

Note that the property (3) in the primitive case is also preserved. In general, outgoing messages originating from a node of a particular layer are forwarded via the corresponding nodes of its upper layers all the way up to the corresponding backbone node and then transmitted to other regions.

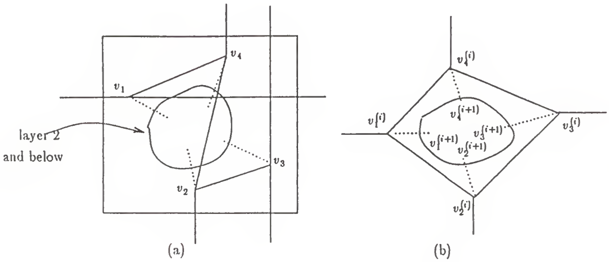

Figure 2.3.2 Backbone Layer and Intermediate Layer Topologies

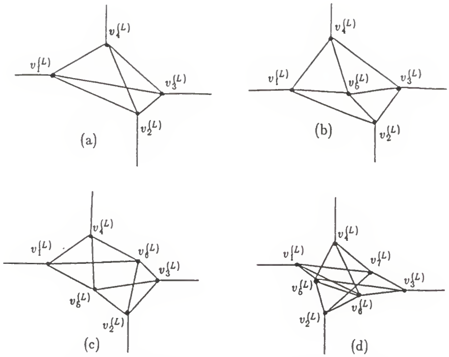

Figure 2.3.3 The Terminal Cases (Final Layer Topologies)

2.3.3 Topologies for Inter-region Links

The inter-region backbone links are established in the second-level task. Only backbone nodes are involved in this task level. Generally, each region will have 4 backbone nodes. To illustrate how inter-region links are established, consider a region **R**, denote the backbone node closest to the west border by v_1, the backbone node closest to the south border by v_2, the backbone node closest to the east border by v_3, and the backbone node closest to the north border by v_4. Let R_i, i $\in$ {1, 2, 3, 4} be the neighboring region of **R**, e.g., let R_1 be the west neighboring region and R_2 be the south neighboring region, . . . etc.; and the backbone node closest to R_i's west border is represented as $v_{i,1}$, . . . etc. See examples in Figure 2.3.3.

The inter-region links are set up based on the following rules:

(1) v_1 is connected to $v_{1,1}$ and $v_{3,1}$, i.e., "horizontal links" are established (since the positions of the nodes change over time, the links are not strictly horizontal, but are so called for the sake of convenience).

(2) v_3 is connected to $v_{2,3}$ and $v_{4,3}$, i.e., "vertical links" are established,

(3) v_2 is linked to $v_{2,4}$ and v_4 is linked to $v_{4,2}$,

(4) If the number of backbone nodes in **R** is less than 4, then some backbone nodes have to play the roles of more than one node.

For example, in Figure 2.3.3.a **R** contains only 2 nodes, while in Figure 2.3.3.b **R** contains at least 4 nodes. Therefore, v_1 and v_2 in Figure 2.3.3.a are performing the roles of v_1 and v_2, and v_3 and v_4, respectively, in Figure 2.3.3.b.

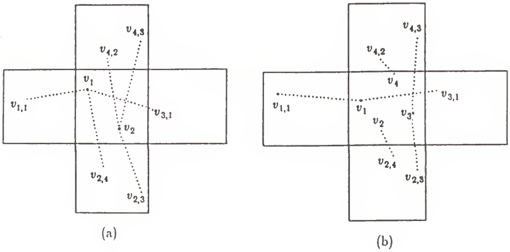

Figure 2.3.3 Examples of Inter-Region Topologies

2.3.4 The Link Assignment Algorithm and Its Analysis

The link assignment algorithm consists of procedures REGION_LINK and NETWORK_LINK. REGION_LINK handles the first-level task and NETWORK_LINK deals with the second level task. It is assumed that the space is partitioned and thus the region borders are determined before the network is initialized. Each node is associated with an ID. To execute the algorithm, each node should have the positional data, represented as (ID, θ, ψ), of all the nodes. The data are obtained through calculations of orbits. In fact, the data can be precomputed, stored in each node and accessed during reconfiguration. With this data, the membership of the nodes (i.e., to which region a node belongs) can be determined during the reconfiguration, and they remain unchanged until the next reconfiguration. If a node is detected dead or missing, the information will be broadcast across the network and the node will be ignored for the reconfiguration. The procedures are described as follows:

```
Procedure REGION_LINK;
Input: a set of nodes of a region of the form-{ID, ($\theta$, $\psi$)}
Steps:
      1. i ← 1;
         If # of nodes < 8, then construct the topology
         according to the arrangement of Figure 2.3.1; return. /* primitive cases */
      2. Construct 2 bi-directional queues from the nodes, $Q_\psi$ and $Q_\theta$;
         $Q_\psi$ is ordered non-decreasingly according to the distance
         between a node and the eastern border;
         $Q_\theta$ is ordered non-decreasingly according to the distance
         between a node and the northern border;
         ties are broken arbitrarily. /* recursive cases */
      3. Remove the first and the last "unremoved" nodes from $Q_\psi$ and $Q_\theta$
         separately. These 4 nodes, marked "removed", form layer i.
         If i = 1, then construct links as specified in Figure 2.3.2.a
             (the backbone layer intra-region topology),
         else construct links as specified in Figure 2.3.2.b
             (the intermediate layer topology).
         i ← i + 1.
      4. If # of "unremoved" nodes in each queue > 7, then go to Step 3.
      5. Construct final layer topology as specified in Figure 2.3.3.
end REGION_LINK.
```

Let R be the region in which the node executing the algorithm is located.

```
Procedure NETWORK_LINK;
Input: the set of nodes of the network of the form-{ID, ($\theta$, $\psi$)}
Steps:
      1. Call REGION_LINK 5 times to derive the regional topologies of
         R, $R_1$, $R_2$, $R_3$, $R_4$.
      2. Construct inter-region topology according to the 4 simple rules
         specified in section 3.3 (examples are shown in Figure 2.3.3).
end NETWORK_LINK.
```

The analysis of the time complexity of the link assignment algorithm is straightforward. Let k be the number of nodes in a region. In REGION_LINK, step 1 and 5 take $O(1)$ (note that the process of checking whether a node is "removed" or "unremoved" in a queue takes $O(1)$ as it can be done by indexing a binary array REMOVED[k]). Step 2, mainly the sorting procedures, take $O(k \log k)$. Steps 3 and 4 serve as a loop. Since 4 nodes are handled each time, the number of loopings is approximately $k/4$. Hence, steps 3 and 4 jointly take $O(k)$. So, the total time complexity of REGION_LINK is $O(k \log k)$.

In NETWORK_LINK, as each node is only concerned with the topologies of its 4 neighboring regions and its own, the time complexity is still $O(k \log k)$. Now, let n be the total number of nodes in the network and m be the number of partitioned regions. It takes time $O(n)$ to determine the memberships of the nodes. By assuming that the number of nodes in each region is approximately n/m, it can be seen that NETWORK_LINK takes time $O((n/m)\log(n/m))$ which is bounded by $O(n \log n)$. So, the algorithm is completed in $O(n \log n)$ time.

2.3.5 Applying the Algorithm

Given that there are 4 transceivers in each node, the maximum node-connectivity of a network is no more than 4. However, consider the case where there exists a region with only one node, In order to maintain a 4-connected network, all the other regions of the same longitude must contain exactly one node. This is simply because there is only one backbone link crossing each of the northern and southern borders for the one-node regions, while there are two such links for the other cases (see Figure 2.3.1). Consequently, a good decision on the size of a region should manage to prevent the one-node-in-a-region case when a network starts its reconfiguration. In fact, we will show that the link assignment algorithm maximizes the node-connectivity of a network if every region of the network contains at least 2 nodes during each moment of reconfiguration. A natural way to achieve this is to increase the region size (i.e., to reduce the total number of regions) such that every region contains at least 2 nodes all the time, which can be done by calculating satellites' orbits and adjusting region sizes accordingly.

The algorithm is most applicable when dealing with a large number of satellites and not very suitable for small satellite networks for two reasons. First, if a satellite network is small, its optimal link assignment can be found by a direct exhaustive search within an acceptable time limit. Second, in order to handle networks in which the number of satellites is small, large region sizes are needed. But, if a region is too large, it will make two nodes within the region invisible to each other; this will make the algorithm fail.

In dealing with large satellite networks, the algorithm can handle the recursive case fairly efficiently. It can take advantage of the locality of nodes and come up with relatively short links (in a dynamic sense, of course, as the length of a link varies) except perhaps for the backbone links. With these short links in a regional network, it is expected that the mean propagation delay for local messages can be significantly reduced. Considering that propagation delay is a substantial part of the end-to-end delay in satellite communications, the algorithm will help speed up the routing in addition to providing high network connectivity.

2.4. Proofs on Network Connectivity

Connectivity is one of the important criteria in network topology design. A network should be able to remain connected even if some links or nodes fail. Before discussing the connectivity of a network, we formally define some terms.

Definition 2.4.1 Let G=(V, E) be an undirected graph representing a network N. V is the set of nodes in N, and E is the set of edges/links in N. A *path* from x to y is a sequence of edges of the form $<v_0, v_1, v_2, ... , v_{j-1}, v_j>$ where (v_{i-1}, v_i) is an edge for $1 \le i \le j$, and $x = v_0$, $y = v_j$. Two paths from v_i to v_j are *node-disjoint* if they contain no common nodes except the two end nodes v_i and v_j. The *node-connectivity* of a pair of nodes is the minimum number of nodes whose removals will disconnect the two nodes. Equivalently, it can be defined as the maximum number of node-disjoint paths between them. The node connectivity of a network is defined as the minimum *node-connectivity* of all pairs of nodes in V. There is another type of connectivity for a graph, called *edge-connectivity* which can be defined as the minimum number of edges whose removals will disconnect a graph. Since node-connectivity is a stronger condition than edge-connectivity in terms of reliability, node-connectivity is adopted here; throughout this chapter connectivity will mean node-connectivity.

Given that each node has at most 4 links in the network, we will show that the link assignment algorithm NETWORK_LINK yields a surprisingly good connectivity in satellite network topology designs.

Definition 2.4.2 Two regions R_i and R_j are adjacent if R_i is contiguous to R_j, or if they intersect in a pole and are in radially symmetric positions with respect to the pole. A *region path* between two regions is a sequence of regions of the form $<R_1, R_2, R_3, R_4, \dots, R_{i-1}, R_i>$ where R_{j-1} is adjacent to R_j, $2 \le j \le i$. Two region paths from R_i to R_j are *region-disjoint* if they have no regions in common except R_i and R_j.

Lemma 2.4.1 Let R = $\{R_1, R_2, \dots, R_m\}$ be the set of the partitioned regions. Let $R_{i,j}$ denote the region [i, i+1] $\times$ [j, j+1], and $R_{i,j} \in$ R. For any pair of regions, there exist 4 region-disjoint paths between them.

Proof. The proof is straightforward. We can show that by explicitly setting up 4 region-disjoint paths between any pair of regions. There are two possible cases to be discussed as follows:

Case 1. $R_{i,j}$ is adjacent to $R_{j,j}$. It is trivially true (see Figure 2.4.1.a for example).

Case 2. $R_{i,i}$ is nonadjacent to $R_{j,j}$. Without loss of generality, suppose that i < j as shown in Figure 2.4.1.b. Construct a path P_1 as $<R_{i,i}, R_{i+1,i}, R_{i+2,i}, \dots, R_{j,i}, R_{j,i+1}, R_{j,i+2}, \dots, R_{j,j-1}, R_{j,j}>$. Construct the second path P_2 as $<R_{i,i}, R_{i,i-1},$

$R_{i+1,i-1}$, ... , $R_{j+1,i-1}$, $R_{j+1,i}$, ... , $R_{j+1,j}$, $R_{j,j}$>. Construct the third path P_3 as <$R_{i,i}$, $R_{i,i+1}$, $R_{i,i+2}$, ... , $R_{i,j}$, $R_{i+1,j}$, ... , $R_{j-1,j}$, $R_{j,j}$>. Construct P_4 as <$R_{i,i}$, $R_{i-1,i}$, $R_{i-1,i+1}$, ... , $R_{i-1,j+1}$, $R_{i,j+1}$, $R_{i+1,j+1}$, ... , $R_{j,j+1}$, $R_{j,j}$>. It is easy to observe that P_1, P_2, P_3, and P_4 are region-disjoint.

The other cases such as $i \geq j$ can be demonstrated similarly and thus are omitted here. □

Each time REGION_LINK is completed, an intra-region topology is constructed with layers of nodes. Except for the final layer, which has 4 to 7 nodes, each layer consists of 4 nodes of the form $\{v_1^{(i)}, v_2^{(i)}, v_3^{(i)}, v_4^{(i)}\}$. For convenience we may call $\{v_1^{(i)}, v_2^{(i)}, v_3^{(i)}, v_4^{(i)}\}$ layer i. In particular, layer 1 is the backbone layer, and also denoted as $\{v_1, v_2, v_3, v_4\}$. Notice that the links, $(v_i^{(j)}, v_i^{(j+1)})$, $(v_4^{(j)}, v_1^{(j+1)})$ and $(v_2^{(j)}, v_3^{(j+1)})$ for all $1 \leq i \leq 4$ and $j \geq 1$, will be assigned in REGION_LINK.

Lemma 2.4.2 There exist disjoint paths from any node x to the other 3 nodes in the same layer i such that these 3 paths consist of nodes only in layer i, layer (i-1), or outside the region.

Proof. There are three cases to be discussed.

Case 1. Here i is the backbone layer. Without loss of generality, suppose $x = v_1$. Let $v_{i,X}$ denote the node v_i in region X(R), where $i \in \{1, 2, 3, 4\}$ and $X \in \{W, NW, N, NE, E, SE, S, SW\}$. Then the 3 disjoint paths are: <v_1, v_4>, <v_1, $v_{1,W(R)}$, $v_{4,W(R)}$, $v_{2,NW(R)}$, $v_{4,NW(R)}$, $v_{1,NW(R)}$, $v_{1,N(R)}$, $v_{4,N(R)}$, $v_{2,N(R)}$, $v_{3,N(R)}$, v_3>, and <v_1, $v_{1,E(R)}$, $v_{4,E(R)}$, $v_{2,E(R)}$, $v_{4,SE(R)}$, $v_{1,SE(R)}$, $v_{1,S(R)}$, $v_{4,S(R)}$, v_2> (see Figure 2.4.2.a). Similarly, we can show that the lemma holds for $x = v_2$, v_3, or v_4.

Case 2. In this instance, i is an intermediate layer. Again, assume that $x = v_1^{(i)}$. Since $(v_1^{(i)}, v_2^{(i)})$ and $(v_1^{(i)}, v_4^{(i)})$ are 2 edges in layer i, 2 paths, <$v_1^{(i)}$, $v_2^{(i)}$> and <$v_1^{(i)}$, $v_4^{(i)}$>, exist. The third path is from $v_1^{(i)}$ to $v_3^{(i)}$, which can be constructed one layer above (i-1). If (i-1) is the backbone layer, i.e., i=2, then the path = <$v_1^{(i)}$, $v_1^{(1)}$, $v_4^{(1)}$, $v_2^{(1)}$, $v_3^{(1)}$, $v_3^{(i)}$> (see Figure 2.4.2.b). If (i-1) is an intermediate layer, the path can be <$v_1^{(i)}$, $v_1^{(i-1)}$, $v_4^{(i-1)}$, $v_3^{(i-1)}$, $v_3^{(i)}$> (see Figure 2.4.2.c).

Case 3. In the third case, i is the final layer. There are 4 topologies for the final layer. In the 4-node topology as shown in Figure 2.3.3.a, there is a complete graph. So, there are 3 disjoint paths from any node to the other 3 nodes. For the 5-node, 6-node, and 7-node cases, it can be checked easily that 3 disjoint paths exist from any node to the other 3, based on the similar enumerative method as in the intermediate case. □

Lemma 2.4.3 There are disjoint paths from any node x to the other 3 nodes in the same layer i, where layer i an intermediate layer, such that these 3 paths consist of nodes only in layer i or i+1.

Proof. Let x be $v_1^{(i)}$. Two obvious disjoint paths are <$v_1^{(i)}$, $v_2^{(i)}$> and <$v_1^{(i)}$, $v_4^{(i)}$>. The third path, from $v_1^{(i)}$ to $v_3^{(i)}$, can be set up via nodes in layer i+1, i.e., <$v_1^{(i)}$, $v_1^{(i+1)}$, $v_4^{(i+1)}$, $v_3^{(i+1)}$, $v_3^{(i)}$>. Since the link assignment in the intermediate layer is symmetric, it holds for the cases where $x \in \{v_2^{(i)}, v_3^{(i)}, v_4^{(i)}\}$. □

Lemma 2.4.4 There are 4 disjoint paths between every pair of nodes $\{x, y\}$ where x and y are in different layers.

Proof. Without loss of generality, suppose that x is in layer i and y is in layer j, i < j. From Lemma 4.2 we know that there exist 3 disjoint paths from x to the other three nodes in layer i by using nodes only in layer i or above (possibly nodes in neighboring regions). Now, since there is an edge ($v_k^{(i)}$, $v_k^{(i+1)}$) for all k ∈ {1, 2, 3, 4} and i is either a backbone layer or an intermediate layer, 4 paths, <$v_k^{(i)}$, $v_k^{(i+1)}$, ... , $v_k^{(j-1)}$, $v_k^{(j)}$>, where k = 1, 2, 3, 4, can be constructed. Hence, we are able to build up 4 disjoint paths from x to $v_k^{(j)}$ for k = 1, 2, 3, 4.

If layer j is an intermediate layer, then by Lemma 4.3 there must exist 3 disjoint paths from y to the other 3 nodes in $\{v_1^{(j)}, v_2^{(j)}, v_3^{(j)}, v_4^{(j)}\}$ that do not contain nodes above layer j. Therefore, there are 4 disjoint paths from x to y (see Figure 2.4.4.a).

If layer j is the final layer (i.e. layer j is equal to layer L in Figure 2.3.3), there are 2 cases. If $y \in \{v_1^{(j)}, v_2^{(j)}, v_3^{(j)}, v_4^{(j)}\} = \{v_1^{(L)}, v_2^{(L)}, v_3^{(L)}, v_4^{(L)}\}$, from the final layer topologies as shown in Figure 2.3.3, it can be seen that each node in $\{v_1^{(L)}, v_2^{(L)}, v_3^{(L)}, v_4^{(L)}\}$ has disjoint paths to the other 3 nodes in the set. So 4 disjoint paths can be constructed from x to y. If $y \in \{v_5^{(L)}, v_6^{(L)}, v_7^{(L)}\}$, since each node in $\{v_5^{(L)}, v_6^{(L)}, v_7^{(L)}\}$ has 4 edges to all of the nodes in $\{v_1^{(L)}, v_2^{(L)}, v_3^{(L)}, v_4^{(L)}\}$ as shown in topologies for terminal cases (see Figure 2.3.3 and Figure 2.4.4.b).
□

Lemma 2.4.5 The connectivity of every pair of nodes in the same layer i is 4.

Proof. It suffices to prove that, for each pair of nodes in the same layer, there exist 4 disjoint paths between them. This can be done by path enumeration. Since it will be a lengthy proof if we enumerate all of the 4 disjoint paths for every pair of nodes, some typical cases are illustrated to show that the disjoint paths can be constructed explicitly. The rest of the cases can be done similarly.

Case 1. Here i is the backbone layer (i = 1). Let us show that, for $(v_1^{(1)}, v_2^{(1)})$, we can explicitly set up 4 disjoint paths connecting them as follows: The first path is $<v_1^{(1)}, v_4^{(1)}, v_2^{(1)}>$. Now, via its west, northwest and north neighboring regions, then going through v_3, a second path can be established. Third, via east, southeast and south neighboring regions, the third path can be set up. Finally, the last path can visit $v_1^{(2)}$, $v_2^{(2)}$, and then $v_2^{(1)}$ (see Figure 4.5.a). It is easy to verify that there exist 4 node-disjoint paths between any other pair in the backbone layer.

Case 2. In this instance, i is the final layer (i = L, see Figure 2.3.3). If the number of nodes in the final layer is 4 (as shown in Figure 2.3.3.a), every pair of nodes is directly connected, so the node-connectivity is a maximum. If the number of nodes in the final layer is 5 (as shown in Figure 2.3.3.b), then, for $\{v_1^{(L)}, v_3^{(L)}\}$ in Figure 2.3.3.b, the 3 disjoint paths between $v_1^{(L)}$ and $v_3^{(L)}$ are $<v_1^{(L)}, v_2^{(L)}, v_3^{(L)}>$, $<v_1^{(L)}, v_4^{(L)}, v_3^{(L)}>$, and $<v_1^{(L)}, v_5^{(L)}, v_3^{(l)}>$.The fourth path will pass through $v_1^{(L-1)}$, $v_4^{(L-1)}$, and $v_3^{(L-1)}$ to $v_3^{(L)}$. Following the same enumerative process, we can verify that it also holds for the 6-node case (as shown in Figure 2.3.3.c) and the 7-node case (as shown in Figure 2.3.2.d).

Case 3. In the third case, i is an intermediate layer. As Figure 2.3.2.b shows, we can choose any pair, say $\{v_1^{(i)}, v_3^{(i)}\}$, and construct the following 4 disjoint paths between them: $<v_1^{(i)}, v_2^{(i)}, v_3^{(i)}>$, $<v_1^{(i)}, v_4^{(i)}, v_3^{(i)}>$, $<v_1^{(i)}, v_1^{(i-1)}, v_2^{(i-1)}, v_3^{(i-1)}, v_3^{(i)}>$, and $<v_1^{(i)}, v_1^{(i+1)}, v_2^{(i+1)}, v_3^{(i+1)}, v_3^{(i)}>$ (see Figure 2.4.5.b). The other cases for an intermediate layer are similar. □

Let $N = (V, E)$ be the network topology derived from applying the NETWORK_LINK algorithm; and let $\{V_1, V_2, \dots, V_m\}$ be the set of the nodes of the regions where each V_i resides in region R_i.

Theorem 2.4.1 If $|V_i| \geq 2$ for all V_i, then the network N has connectivity = 4.

Proof. Let v_i and v_j be two nodes in V.

Case 1. Here v_i and v_j belong to the same region R_i. If R_i contains fewer than 8 nodes, i.e., a primitive case (Figure 2.3.1.b to Figure 2.3.1.f), we can enumerate all the 4 disjoint paths between any two nodes. These paths will visit 4 adjacent regions of R_i (i.e., N(R_i), S(R_i), W(R_i), and E(R_i)), and 4 non-adjacent regions (i.e., NW(R_i), NE(R_i), SE(R_i), and SW(R_i)). For the sake of briefness, they are not enumerated here (a similar example is shown in Figure 2.4.5.a). If R_i contains more than 7 nodes, it follows immediately from Lemma 2.4.4 and Lemma 2.4.5 that 4 disjoint paths exist for each pair.

Case 2. In the second case, v_i and v_j are not in the same region, (e.g., v_i is in R_i and v_j is in R_j). If R_i is reconfigured as a primitive topology, it is easy to verify that there exist 4 disjoint paths from any node in any of the 7 primitive cases to its 4 adjacent neighboring regions. If R_i contains more than 7 nodes, then as we have shown in the proof of Lemma 2.4.2 and Lemma 2.4.3, there are 4 disjoint paths from v_i to the four backbone nodes of R_i; and each backbone node has an edge to its neighboring regions. Thus, regardless of the number of nodes R_i gets, there are always 4 disjoint paths from v_i to its 4 neighboring regions. Similarly, there are always 4 disjoint paths from v_j in R_j to its 4 neighboring regions. Now, by applying Lemma 2.4.1 we can construct 4 disjoint region-paths from v_i to 4 of its neighboring regions. Finally, via these disjoint region-paths 4 disjoint paths from v_i to v_j can be successfully established. □

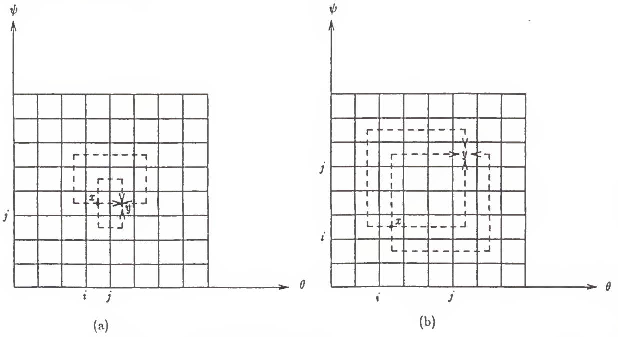

Figure 2.4.1 Disjoint Region-paths

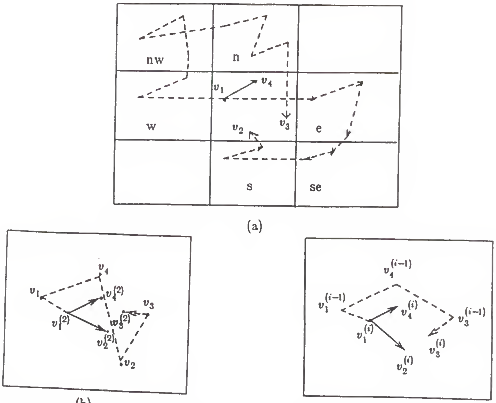

Figure 2.4.2 Three cases of Lemma 3.4.2

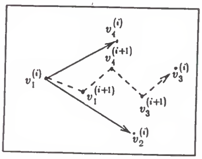

Figure 2.4.3 Routing in the same layer

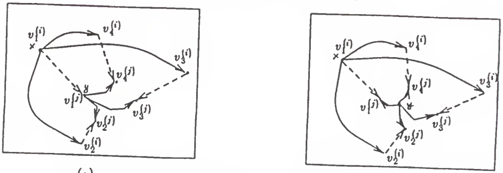

Figure 2.4.4 Routing in different layer

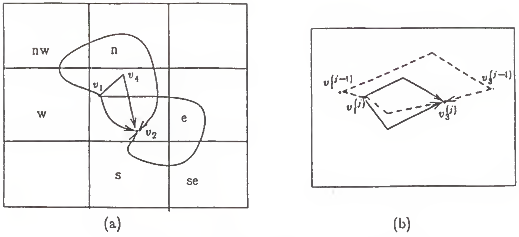

Figure 2.4.5 Two examples of external routing

2.5 Failures and Recoveries

In a mobile satellite network, a broken link can be detected by the loss of signal, which may be due to the failure of a transceiver on either side. The failure of a node is detected through a consensus reached by its adjacent nodes. With 4-connectivity, a network can survive without being partitioned as long as no more than 4 nodes or links fail. Yet, there are backbone links and nodes whose failures have more serious impacts on the overall performance of a network than those of non-backbone links and nodes. When a backbone node/link fails, a local topological reconfiguration is needed to maintain sufficient backbone links and nodes; non-backbone link/node failures may be handled in the routing level (e.g., by reconstructing virtual circuits) instead of in the link assignment level. As failure events occur asynchronously, the recovery should be done dynamically. The recovery will require each party involved to share the same information about the local reconfiguration and will need a centralized algorithm distributively run by the affected nodes. This centralized recovery algorithm will do link-switching if a backbone link fails. It will also do backbone-node-selecting and link-switching if a backbone node fails.

It is assumed that all the failure events are single-failure events; a multiplefailure event rarely occurs. When it does occur, its failures are treated just like a sequence of single failures. One thing we would like to point out is that, if any failure occurs at a time very close to the next reconfiguration, the recovery algorithm may ignore it because it will soon be taken care of by the global reconfiguration.

2.5.1 Link Failures

Failure on a non-backbone link will be ignored as the network has a high degree of connectivity. A failure on a backbone link will not partition the network, but it will have a significant impact on the routing. All packets going to other regions are supposed to be forwarded through one of the backbone links. A failed backbone link will cause packets to be routed via longer paths and may create traffic jams for other backbone links. Hence, link failure of this kind should be handled properly. This can be done through link-switching. The backbone node which has a malfunctioning trans-ceiver will consult the other three nodes connected to it and decide which of the three links will be used to replace the broken backbone link. In this way the network can cushion the impact of a broken link to the minimum.

2.5.2 Node Failures

A node may be detected to have failed (e.g., through the failures of all the links incident to it). The failure of a non-backbone node will not result in a local reconfiguration. But a backbone-node failure needs to be handled properly, considering its more important role. The recovery process works as follows: The region containing the failed node will select a node to replace it. The principle for the replacement is "choosing-the-nearest-node," i.e., the node that is closest to the failed node is chosen as the new backbone. As each node has information on the locations of the other nodes, this can be done in a straightforward manner. By doing this, those nodes originally linked to the failed node will be able to connect to the new backbone node.

CHAPTER III
RELIABLE MESSAGE BROADCAST

3.1 Background

For various reasons, a node in a network needs to broadcast messages to all other nodes. A broadcast protocol is reliable if every node can receive the broadcast messages in finite time. In networks with fixed topologies, broadcasting is usually done by delivering messages through some broadcasting trees (e.g., a minimum-spanning tree) [7,10,11,14,15,26-28] for the sake of reducing the number of messages exchanged. For networks where the topology keeps changing, the task is more complicated because a pre-constructed routing path may be disconnected. Even if no paths are disconnected, with varying propagation delays over communication links, the routing of the broadcast messages may not be optimal if pre-computed broadcasting paths are used in the protocol.

It has been shown by Awerbuch and Even [3] that a reliable broadcast in a changing-topology network is possible if the network is "*eventually connected*" (no nodes are permanently isolated from the others). Some sophisticated broadcasting algorithms have been developed to achieve reliable broadcast in eventually connected networks with slowly changing topologies [3][32][35]. However, the eventually connected networks for which these broadcast algorithms were designed are mainly stationary networks undergoing arbitrary link failures and recoveries and thus the significance of these algorithms are very restrictive. In addition, the algorithms are not guaranteed to achieve minimum broadcast delay in networks with rapidly changing topologies such as mobile communication networks described below.

In this research, the concept of eventually connected networks is further extended to include mobile networks subject to frequent link reconfigurations [30]. Typical networks of this kind include large-scale mobile communication networks such as packet radio networks and point-to-point satellite networks. There are three fundamental properties in the topologies of the networks: (i) each node is associated with a number of communication ports with which communication links can be established, (ii) the nodes are mobile and communication delay across each link is varying, (iii) link reconfiguration is necessary in order to preserve the connectivity of the network [7][17-18][22]; this is due to restrictions such as limited transmission powers in the transceivers of packet radio networks when nodes are too far apart, or the blocking of the earth in satellite networks connected by laser transceiver.

It has been shown [26] that in changing-topology networks, if a broadcast message is not buffered, it is likely that some node may never receive that message. For example, suppose that i is the only node connected to j and a message M arrives at i immediately after the link between i and j is broken, if j later reconnects to i or connects to any node which has already received M and discarded it, j will not receive M.

The questions which arise are: 1) how long a message should be buffered and 2) how should the message buffers be managed. These issues have never been successfully addressed in networks with changing topologies. In [3], the algorithms work by assuming infinite message buffer space in each node. However, in any network each node has only a finite memory. In this research, we show that bounded message buffers are feasible for a reliable broadcast in eventually connected networks.

Toward this end, we take a rather unconventional approach to the design of reliable broadcast protocols. By using message reply vectors, we design a reliable and efficient broadcast protocol which requires bounded message buffers for broadcast messages in each node. The protocol also provides a mechanism which enables obsolete messages to be dynamically flushed out. This will substantially increase the utilization of limited buffer space. In particular, the algorithm is proven to be optimal in terms of broadcast delay in eventually connected networks with arbitrarily changing topologies.

3.2 Network Model

Let a mobile communication network be represented as G = (V, E), where V represents the nodes in the network, and E represents the operating communication links in the network. $|V|$ is fixed while $|E|$ is bounded and time-varying.

3.2.1 Preliminary Definitions

In addition to all the basic notions of graph theory, we define some related terms as follows. A *directed tree* in a network is a tree in which each node, except for the root of the tree, has a neighboring node (nodes connected by a link are *neighbors*) as its *parent* node, and each parent has one or more *child node*s or *children*. A link is *connected* if it provides at least a minimum operating interval τ during which a message can be successfully transmitted, otherwise it is considered *disconnected*. Two nodes that are connected by a link but that do not have a parent-child relationship are *siblings*. Two trees are called *adjacent* if they are connected by at least one link. The links of a tree are called *parent-child links* while the links by which two siblings are connected are *sibling link*s. The sibling links can be further classified as *internal* and *external*; an internal sibling link connects two siblings in the same tree (i.e., they have the same root), and an external sibling link is an edge between two siblings in different trees. The issue on how the relationship between neighbors is determined will be discussed in the next section. A *path* P from x to y is a sequence of edges of the form $<v_0, v_1, v_2, \ldots, v_{j-1}, v_j>$ where (v_{i-1}, v_i) is a link for $1 \le i \le j$, and $x = v_0$, $y = v_j$. If every (v_{i-1}, v_i) is a parent-child link, then P is called a *primary-path*; if some (v_{i-1}, v_i) are siblings link, then P is called an *alternative-path.*

The concept of *eventually connected* networks is defined as follows. A node u is said to be eventually connected to node S in G if, at any time t_0, there exists a constant β and a node sequence P = $<v_0, v_1, v_2, \ldots, v_{j-1}, v_j>$ where $v_0 = u$ and $v_j = S$, such that a time sequence T = $(t_1, t_2, \ldots, t_j)$ can be found with $t_0 \le t_1 \le t_2 \le \ldots \le t_j \le t_0 + \beta$, and $\{v_{i-1}, v_i\}$ is connected by a link at t_i and a minimum operating interval $[t_i, t_i+\tau]$ exists for $\{v_{i-1}, v_i\}$. If every node in E is eventually connected to node S, then E is said to be eventually connected with respect to node S. The node sequence P above does not have to be a physically connected path (P can be called an *eventually connected path*). Nor is it assumed that a message can always be delivered from u to S via P within the interval $[t_0, t_0 + \beta]$.

3.2.2 Assumptions

It is assumed that the following statements hold:

(1) Each node has a unique ID and has a complete list of the IDs of the other nodes in the network, but has no information about the complete network topology; the topological knowledge of a node is restricted to that of its neighbors (in a network undergoing topological changes, it is impractical to assume that complete topological data is available to every node).

(2) Delivery is reliable across a connected link and the time to transmit a message through a link is finite and bounded.

(3) The time spent in processing an incoming or outgoing message is negligible, compared to the propagation delay of sending a message through a link.

(4) Links may reconfigure for certain reasons (e.g., maintaining network connectivity), which are unpredictable to each node; each reconfiguration may involve disconnections (link-downs), connections (link-ups), or switchings of links (reconfigurations that are not due to a link failure or recovery). In addition, if the reconfiguration is an "involuntary" one, i.e., a link failure, it can be detected

by the nodes on both ends. It is also assumed that a node will never fail so that every node will respond to an incoming message.

(5) The network is eventually connected with β bounded; an unbounded β would imply that some nodes may be isolated from the rest of the network for an arbitrarily long time regardless of what protocol is used, which has little or no practical significance in message broadcasting. Observe that this assumption does not imply that every link reconfiguration should be finished in finite time or that every failed link should be recovered in finite time. All it requires is that the network be maintained in such a way that an eventually connected path exists between any two nodes.

3.3 The Two-Phase Broadcast Protocol

3.3.1 Description of the Protocol

The broadcast protocol is a distributed algorithm in the sense that each node reacts to different conditions independently. For the purpose of brevity and ease of demonstration, a single-source broadcast environment is assumed. By assuming that broadcast messages do not interfere with each other, and that each node treats the broadcast messages from different sources independently, the protocol can be easily extended to multiple-source cases. Each broadcast message is associated with an acknowledgment message or a reply message that, appended on the end of the broadcast message, contains one bit position for each node in the network (i.e., an n-bit vector initialized to 0). Throughout this chapter, we use reply messages (or acknowledgment messages) and reply vectors (or acknowledgment vectors) interchangeably. For the following discussion of the protocol, we shall assume that the broadcast messages are from a single source node S. The protocol consists of two phases: broadcast phase and reply phase.

3.3.1.1 Broadcast phase

In the broadcast phase, each node performs the flooding procedure [33][35], i.e., each node receiving an incoming message M will keep a copy and send it to all the neighboring nodes except the nodes from which M came. Unlike conventional floodings, a message received is buffered in memory and will not be discarded until an explicit *flush-out* message from S arrives. The broadcast phase begins with S sending a message to each of its neighbors. Let node i be a node that has just received M, and $NEIB_i$ be the set of neighboring nodes of node i. When node i receives M, it sets the position corresponding to i in the reply vector of M to 1. The first node from which M arrives is chosen to be the parent of i, denoted by PR_i, and a parent-notifying control message is sent to PR_i acknowledging the relationship. The nodes from which node i receives a parent-notifying message form the set of node i's children, denoted by CR_i. The other nodes sending M to i are the siblings of i, denoted by SB_i. Again, a control message is sent to each sibling to acknowledge the relationship. Notice that SB_i consists of either the nodes which exchange the same M with node i due to the propagation delay, or the nodes which deliver M to i slightly later than PR_i. Siblings play an important role in the reply phase when a parent-child link is disconnected.

During the broadcast phase, if a new link is established between two nodes, both sides will exchange information about broadcast messages. Each node will transmit over the link the messages the other side does not have. As the broadcast messages are spreading across the network, a spanning tree rooted at S is constructed and expanded. The spanning tree consists of parent-child links and will be used to route the reply messages of M. If the network is connected and no links are disconnected during the broadcast phase, a complete spanning tree can eventually be established. If links are broken in the meantime, then the spanning tree could be partitioned into two or more subtrees, which may cause reply vectors to be detoured.

Theorem 3.1 The broadcast protocol is reliable if the network is eventually connected.

Proof. Let M be a broadcast message from S. Assume that v never receives M. Since the network is eventually connected and M is broadcast by flooding, the number of nodes that receive M must increase over a finite period of time. If v never receives M, then the number of nodes receiving M must stop increasing after a certain time, and this number is less than $|V|$. Let C_1 be the set of nodes receiving M and $C_2 = V \setminus C_1$ where $v \in C_2$. As the protocol ensures that each node will inform its neighbors of M, there must be a permanent edge-cut between C_1 and C_2. This implies that the network is not eventually connected, a contradiction. Hence, the broadcast protocol is reliable. $\square$

3.3.1.2 Reply phase

A node receiving M initiates the reply phase under the following two conditions: 1) it has no neighbors to which M needs to be sent or 2) all the edges to its children are disconnected. In the former case, the broadcast phase ends at this node and we call such node a leaf node having no children. In the latter case, there is no need to wait for the reply messages from its children as it has no way of knowing when or whether the links connected to the children will be restored. In both cases, these nodes are called initiating nodes.

If a node is not an initiating node, it accumulates all the vectors acknowledging the acceptance of M from its connected children, logically ORs them together to form a new version of the vector, and forwards the copy of this vector to its own parent node. Notice that, by piecing together the reply vectors from its children, a node can also construct a descendent list of its own.

Lemma 3.1 If each parent-child link is not disconnected before the child node finishes replying its vector, then every reply message can be sent to S through a primary-path.

Proof. If each parent-child link is not disconnected before the child node finishes replying its vector, then each node can reply to its parent via the parent-child link. Hence, the reply message will be forwarded to S through a path consisting of only parent-child links, which is a primary-path. $\square$

During the reply phase, if a node j is disconnected from PR_j, a sibling in SB_j is chosen as the new parent of j. If it happens that two siblings are disconnected from their parents and thus acknowledge each other as their new parents, the tie is broken arbitrarily so that one is the parent and the other is a child. If node j has no siblings, a new tree, splitting from the old, emerges with j being the root. This may result in an alternative-path search if j has not yet replied its vector, because no primary-paths connecting j and S exist.

The alternative-path algorithm. The alternative-path algorithm is based on the idea of finding external siblings. When a node, which has lost its parent and has no connected siblings, is unable to send a reply vector, it starts alternative-path search by passing the vector to its children. Receiving the vector from its parent, each child updates its own vector and sends a copy of the vector to each of its external siblings. As an internal sibling is a neighboring node in the same tree, it is useless to exchange reply vectors among internal siblings because they lead to no new paths to other trees. An external sibling can be discerned from an internal one from the information contained in the reply vector sent by the parent (i.e., the internal siblings will have 1s' in the reply vector).

In addition to sending the updated reply vector to its external siblings if it has any, a node needs to pass the vector to each of its own children. If a node has no siblings, it simply passes the vector to each of its children. The necessity of sending vectors to children even if external siblings are found is justified by the following lemma.

Lemma 3.2 An alternative-path is not guaranteed to be found if not every node in the new tree, generated as a result of link disconnection, is explored.

Proof. Without loss of generality, suppose that v in the new tree T_i is a leaf node connected to the root r via a path P = $<r, v_1, \dots, v_i, v>$ and that v has no external siblings. Assume that the alternative-path search will only explore the nodes having external siblings. Therefore, the search stops at some node v_j where $1 \leq j \leq i$ and v_j has external siblings. If it happens that none of the external siblings of T_i can lead a path to S but a new link is just established between v and a node u not in T_i and u has a path to S, this path then becomes the only alternative path to S. Since v is unexplored and thus does not receive a vector from its parent, the alternative-path may never be found even if the network is eventually connected. □

Figure 3.1 shows a simple example. Suppose that an alternative-path search will stop when the first external sibling is found. Let r_1 and r_2 be roots of two adjacent trees and v_1 and v_2 be two external siblings. Both r_1 and r_2 start alternative-path search due to the disconnections of the links to their parents. The reply information on both trees may be exchanged over the external sibling link connecting v_1 and v_2, then sent to r_1 and r_2, and finally back to v_1 and v_2 again. Both sides exchange reply vectors again and reject each other as their reply vectors are identical. The search on both sides stops at v_1 and v_2 without discovering that there exists an alternative-path via v_3 to S.

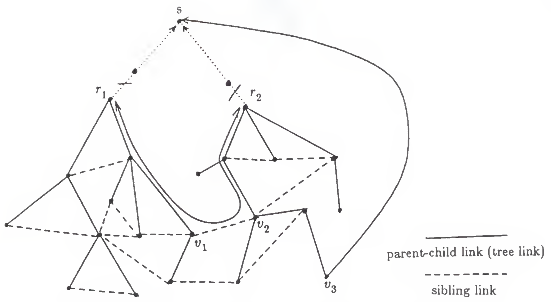

Figure 3.1 A failed alternative-path search

When a sibling node in a neighboring tree receives a reply message via an external sibling link, if the node has already done its own replying and the vector contains new information that it does not have, it will forward this vector immediately to its parent. If it has not yet replied with its own vector or the vector contains new information, it will create an update vector by merging this vector with its own, wait for its children's reply vectors if it has not yet replied, and then reply the vector. If the vector contains no additional reply information, it will be rejected by this external sibling.

It is possible that an external sibling may receive the same reply vector from different external siblings from the same tree. This "multiple-copy-to-a-single-sibling" situation causes redundancy, but it is risky to be avoided completely. The reason is as follows: Suppose that the algorithm can be designed in such a way that no siblings will be sent with more than one copy of the vector. This can be done by the following steps: 1) search in parallel all the nodes in the tree (assuming that the search is not interrupted by link disconnections), 2) bring back information about each external sibling from each descendent, 3) let the tree root decide which external sibling should be sent by which node. However, an alternative-path may be missed if the designated sibling is disconnected from its external sibling after the duty is assigned and before it can send the reply vector. Missing a particular alternative-path may cause a serious delay or a permanent blocking of the vector-replying for reasons similar to Lemma 3.2.

However, the situation can be safely alleviated by refining the algorithm as follows: A list of external siblings which have been reached by the reply vector is sent, along with the vector, to each children so that each node will not send the vector to the siblings on the list. After each node has finished routing the vector, the external siblings just reached with the vector are added to the list, which is then delivered to its own children.

The alternative-path algorithm propagates all the way down to the leaves of the tree. If none of these descendents of the root have external siblings, the tree is temporarily isolated from the rest of the network. As the network is assumed to be eventually connected, in finite time at least one node will connect to the other nodes outside the tree through link reconfiguration. This ensures that the reply vectors can continue to be forwarded toward S. It is likely that, due to the operation of a network reconfiguration algorithm, some link disconnections may result in a partition of the network; a spanning tree becomes a spanning forest. Therefore, an alternative-path eventually leading to S may consist of a sequence of parent-child links and external sibling links.

The sibling-reclassify algorithm. After replying to its parent, a node is still likely to be disconnected from its parent and become a root of a new tree if it has no siblings. This makes the vector reply more complicated. It seems at first that this condition can simply be ignored since the reply vector, which contains reply messages of all the nodes within this tree, has been sent out of this tree. However, ignoring this condition could result in a long delay or permanent blocking for replying vectors. Consider the following example. Let x and y be two nodes in tree T_1 and x is a child node of y. Suppose that x has no siblings. If x is disconnected from y, then x and y belong to different trees with x being the root of a new tree T_2. Now, the sibling links between nodes in T_1 and nodes in T_2 change from internal to external. If these links are not "reactivated" as external links, where it might be the case that the only alternative-path from nodes in T_1 to S must go through nodes in T_2, then the alternative-path can not be found. Therefore, sibling links between T_1 and T_2 need to be reclassified.

The sibling-reclassify algorithm works in a way similar to that of the alternative-path algorithm. The primary difference is that the sibling-reclassify algorithm requires a specific control message, originating from the root of the new tree, to indicate which nodes are still in the same tree. This is because all the nodes in the tree may have the same information in their reply vectors due to the alternative-path search process invoked before the tree is split into two. Hence, a node has no way of knowing who its external siblings by merely referring its reply vector as what have been done in the alternative-path algorithm. The specific message contains the descendent list of the root, which is constructed when the root is doing its first vector reply.

The algorithm proceeds as each node sends to its children the message containing the set of all the nodes in the tree. Each node receiving this message can determine which siblings have become external, and sends each of them a message declaring the relationship to be external. A node which receives such a declaration message will acknowledge the sending node with its reply vector. This is an important step to prevent the other side from missing the new reply information which arrives when the links have not yet been declared to be external.

In summary, when a link is disconnected in the reply phase, the relationship between the nodes on both ends terminates. If a node is disconnected from its parent, it will select a new parent from its siblings if it has any. Otherwise, a new tree is born. If the node has not replied its vector, the reply of its vector has to be done by invoking the alternative-path algorithm. If the node has already finished replying, it reclassifies the siblings so that siblings in neighboring trees can be reactivated and can be used as possible bridges leading to the source. On the other hand, when a new link is established, two nodes on both ends of the link will consider each other as siblings and exchange message status in order to resume message broadcasts and/or to continue vector-replying.

Lemma 3.3 Let C be a connected component and S is not in C. If C remains connected but isolated from the rest of the network sufficiently long, the reply vectors of nodes in C will converge to an identical form.

Proof. We consider the four possible cases.

Case 1: Suppose that no links are disconnected or newly connected in this connected component C for a sufficiently long period of time. If there is only one tree in C, then all the reply vectors will reach the root. Since the root can not forward its reply vector, it will send it to all of its descendents which will update their own reply vectors. This means every node in this tree will have an identical reply vector. If there are more than one tree, say T_1, T_2, ... ,T_m, there must exist at least one external sibling link between any two adjacent trees. Let T_i and T_j be two adjacent trees. Since the roots of T_i and T_j will eventually invoke the alternative-path algorithm, there must be some exchanges of reply information between nodes in T_i and T_j through external sibling links. As each node will route the reply vector from its external sibling in the adjacent tree to its parent, the reply will eventually reach the tree root. The root will soon propagate it to all the nodes in the tree as a result of the alternative-path algorithm. Hence, we can see that nodes in T_i and T_j will have the same reply vectors. Similarly, it can be inferred that all the trees in the same connected component will have the same reply vector if C is isolated long enough.

Case 2: Suppose that some links are disconnected but C remains connected. Without loss of generality, let L_i be the link that is disconnected. There are two cases to be discussed:

Case 2.1: If L_i is a parent-child link, either the child selects a new parent or it produces a newly-born tree. In the former case, if the new parent is in the same tree, it causes no problem because it is only a switching of a parent-child link. If the new parent is not in the same tree (an external sibling is chosen), the split new tree rooted at the child is now merged into another tree; the total number of trees in this component remains the same. As in *Case* 1, the merged tree will eventually share the same reply information as a result of exchanging reply vectors. In the latter case, a new tree is born, which increments the number of trees in this component by 1. The disconnection either invokes the alternative-path algorithm or sibling-reclassify algorithm. In both cases, the external siblings between the new tree and the old tree (i.e., its previous parent's tree) are declared and exchange reply information. Similar arguments as in *Case* 1 can be applied for the rest of this case.

Case 2.2: If L_i is a sibling link, then it does not affect the convergence. The reason is as follows. If L_i is internal, the disconnection only drops out a useless link in the tree. If L_i is external, as C remains connected, even if L_i is the last external sibling link connecting the two trees, the nodes in both trees can exchange reply information via other trees between them. The rest of this case is similar to *Case* 1.

Case 3: Suppose that some links are established. Since a new link will only make nodes on both sides exchange broadcast and reply information; it does not block vector-replying. On the contrary, it may speed up the process of the convergence.

Case 4: Suppose that some links are disconnected and some are connected. Similar reasoning can be applied as in *Case* 2 and 3. □

Lemma 3.4 Each reply vector can be delivered to S through the primary path or an alternative-path in time that is finite and bounded.

Proof. Let v be a node sending a reply message to S. If there is no link disconnection, from Lemma 3.1 we know that S can receive v's reply message. If some links in the primary path from v to S are disconnected, two possible cases are considered:

Case 1: assume that v and S are in the same connected component C for sufficiently long. Let S be in tree T_s and v be in tree T_v. If T_s and T_v are adjacent to each other, then v can be routed to S via a sibling link between the two trees. If T_s and T_v are not adjacent, there must exist some trees $T_1, T_2, \ldots, T_i$, such that T_s is adjacent to T_1, T_1 is adjacent to T_2, ... , and T_i is adjacent to T_v. Without loss of generality, we assume that no new tree is born in the meantime. From the alternative-path algorithm, we know that there exists an alternative-path between T_s and T_v via $T_1, \ldots, T_i$. Since each vector can be sent over a link in finite and bounded time, v's reply vector can be delivered to S in time that is finite and bounded.

Case 2: assume that v and S are not in the same connected component. Let C_v be the set of nodes which have received v's reply information. As the reply process proceeds, $|C_v|$ is increasing like a step-function. From Lemma 3.3, it can be seen that v's reply information will not stop propagating as long as some nodes in the same connected component have yet to receive it. If $|C_v|$ stops increasing temporarily, then there must exist an edge-cut between C_v and $V \setminus C_v$. Since the network is eventually connected, there must exist a node u and a node w such that $u \in C_v$ and $w \in V \setminus C_v$ and a new link $\{u, w\}$ is established in finite and bounded time. Therefore, due to the reply information exchanges between u and w, C_v is now expanded to be $C_v \cup w$. As the total number of nodes in this network is fixed, S will get v's reply message in time that is finite and bounded. Similarly, we can conclude that every reply vector will be received by S in finite and bounded time. □

When the source S receives all of the reply vectors for message M (i.e., the final version of the vector has value 1 in every bit in the vector), it attaches a flush-out message in its next broadcast message ready to be sent, which directs each node receiving this message to discard M. A flush-out message is considered as a part of the broadcast message to which it is appended and it will be discarded along with the broadcast message later.

It is important to note that the protocol does not imply that each broadcast message can not be broadcast until all the reply information of its predecessor has been received by S. As indicated earlier, each node can treat each broadcast message independently. If more than one broadcast message from the same source are allowed to coexist in the network, what has to be done is to allow each broadcast has its own "family" in each node. As the network is subject to frequent link reconfigurations and propagation delay in each link is changing, a broadcast message should not take advantage of the direct tree established by its predecessor by broadcasting along the tree links. Flooding is still needed if reliability and minimum broadcast delay are to be achieved.

Theorem 3.2 Every broadcast message will be discarded in finite and bounded time.

Proof. From Theorem 3.1 and Lemma 3.4, we know that the source S will detect that every broadcast message has received a broadcast message in finite and bounded time. Hence, a flush-out message can be broadcast to clear that message, which implies that the stay of each broadcast message in each node is finite and bounded. □

Corollary 3.1 If the generation rate of broadcast messages is bounded, then the buffer space needed in each node is bounded. □

3.3.2 Message Buffer Management

Message buffer management is one of the major issues in communication protocol designs. As each node has only limited memory space, this issue becomes critical because an ineffective management policy can result in a great waste of buffer space or, even worse, an unreliable protocol. Theoretically, the broadcast algorithm above can still work as a fast and reliable communication protocol without the reply phase, provided that each node has a sufficiently large memory to buffer broadcast messages and is capable of determining the right time to discard the messages. Without a reply mechanism, there are two possible ways of managing message buffers:

(1) *Static* policy - Each message received will remain in the buffer only a fixed period of time and then be eliminated from the buffer. This fixed interval is determined by an upper bound which is long enough to ensure that a message has been received by every node. The weakness of the method is two-fold: First, it is very difficult to accurately estimate the upper time limit for a broadcast message to stay in the buffer and the limit tends to be over-estimated. Second, by fixing the time a message has to stay, it is possible that too many obsolete messages, i.e., messages already received by every node, stay in the buffer memory for too long. This will cause a significant waste of the buffer space.

(2) *Forced-out* policy - Broadcast messages are discarded only when the buffer is full and messages are discarded in a first-in-first-out order. This method has the same weakness as the static policy in wasting buffer space. In addition, it also causes great complexity in allocating message buffers since broadcast messages are not the only messages a node needs to handle. If a broadcast message buffer

is too small, a broadcast message may be eliminated while some nodes are still waiting for that message. This violates the reliability of broadcasting. If too large a broadcast message buffer is allocated, again, memory space is substantially wasted.

The two-phase broadcast protocol provides a dynamic buffer management scheme. This management scheme has two major advantages: First, the reliability of the protocol is achieved in a more flexible manner. Since each message will be discarded only if the corresponding flush-out message is received, a broadcast message is allowed to stay in the buffer as long as it is needed without being accidentally eliminated from the buffer. If few link reconfigurations occur during the broadcast and reply phase, a message can be discarded quickly. If more link disconnections and connections occur, a message is allowed to stay longer. In short, the protocol's dynamic management policy provides a significantly better buffer utilization than the two policies above. Second, Corollary 1 implies that each node can estimate the upper bound of the buffer size needed to store broadcast messages by estimating the maximum time interval a message will stay. With this time bound closely approximated, the buffer allocation problem can be substantially simplified.

3.3.3 The Node Algorithm

In this section we present the two-phase broadcast protocol in the form of a node algorithm, consisting of a number of event-driven procedures. A node reacts to each condition/event by running the corresponding procedure. Before we formally describe the node algorithm, some terms and variables used in the algorithm are defined as follows.

a. M : a broadcast message from the source S.

b. ACK-parent(i): the parent-notifying message from node i to its parent.

c. ACK-sibling(i): the sibling-notifying message from node i to its sibling.

d. DCL-exsib(i): the message from node i to declare external-sibling.

e. ACK-vector(M,i) : the current acknowledgment vector of a broadcast message M at node i. It is an n-bit binary vector and ACK-vector(M,i) $\supseteq$ ACK-vector(M,j) if ACK-vector(M,i)[k] $\geq$ ACK-vector(M,j)[k] for all k, where $1 \leq k \leq n$. ACK-vector(M,i)[k] is 1 if node i "knows" that node k has received M, 0 if otherwise.

f. UNK_i : neighbors of node i and whose relationships to i are still unknown (e.g. they could be either siblings or children of i); UNK_i is initialized to be $NEIB_i$.

g. $E\text{-}SB_i$: the set of i's external siblings.

h. DES_i : the set of i's descendents.

i. $SB\text{-}DONE$: the set of the external siblings already been reached by the same reply vector from ancestors.

j. BUF_i : message buffer of node i.

k. FLUSH-OUT(M): a message indicating that M can be discarded from the buffer memory.

l. $LINK-DOWN_j$: a message indicating that the link to node j is disconnected.

m. $LINK-UP_j$: a message indicating that the link to node j is established.

n. \: difference, a set operation.

o. **or**: a bitwise logical-or operation defined on n-bit ACK-vectors.

```
The Basic Node Algorithm for node $i$:
 For a message M from j:
    a.1)    Mark $j$ RECEIVED for M;
    a.2)    $UNK_i \leftarrow UNK_i \setminus \{j\}$;
    a.3)    if M is in $BUF_i$, $SB_i \leftarrow SB_i \cup j$ and send ACK-sibling($i$) to $j$,
    a.4)    else begin /* $j$ is the first node sending M to $i$ */
    a.5)        put M into $BUF_i$;
    a.6)        ACK-vector(M,$i$)[$i$] $\leftarrow$ 1;
    a.7)        send ACK-parent($i$) to $j$;
    a.8)        $PR_i \leftarrow j$;
    a.9)        $UNK_i \leftarrow NEIB_i \setminus PR_i$;
    a.10)       for every $k \in NEIB_i \setminus \{j\}$,
    a.11)           send M to $k$;
```

```
  a.12)          end
  a.13)       if every $k \in NEIB_i$ is RECEIVED for M and $UNK_i = \varnothing$,
  a.14)          call reply_ack;
For an ACK-parent($j$):
  b.1)      $CR_i \leftarrow CR_i \cup \{j\}$;
  b.2)      $UNK_i \leftarrow UNK_i \setminus \{j\}$;
For an ACK-sibling($j$):
  c.1)      $SB_i \leftarrow SB_i \cup \{j\}$;
  c.2)      $UNK_i \leftarrow UNK_i \setminus \{j\}$;
For a $LINK\text{-}DOWN_j$ message:
  d.1)      $NEIB_i \leftarrow NEIB_i \setminus \{j\}$;
  d.2)      if $j \in UNK_i$, $UNK_i \leftarrow UNK_i \setminus \{j\}$;
  d.3)      if $j \in SB_i$, $SB_i \leftarrow SB_i \setminus \{j\}$;
  d.4)      if $j \in E\text{-}SB_i$, $E\text{-}SB_i \leftarrow E\text{-}SB_i \setminus \{j\}$;
  d.5)      if $j \in CR_i$,
  d.6)      begin
  d.7)          $CR_i \leftarrow CR_i \setminus \{j\}$;
  d.8)          if $i$ is not REPLIED and ($CR_i = \varnothing$ or every $k \in CR_i$ is REPLIED),
  d.9)              call reply_ack;
  d.10)      end
  d.11)      if $j = PR_i$ and $SB_i \neq \varnothing$,
  d.12)      begin
  d.13)          choose a new parent $k$ from $SB_i$;
  d.14)          send ACK-parent($i$) to $k$;
  d.15)      end
  d.16)      else if $j = PR_i$ and $i$ is REPLIED,
  d.17)          begin /* starts sibling-reclassifying */
  d.18)              create $DES_i$;
  d.19)              send $DES_i$ to every $k \in CR_i$;
  d.20)          end
For a $LINK\text{-}UP_j$ message:
  e.1)      $NEIB_i \leftarrow NEIB_i \cup \{j\}$;
  e.2)      $SB_i \leftarrow SB_i \cup \{j\}$;
  e.3)      $i$ exchanges broadcast message status with $j$;
  e.4)      for every M not received by $j$,
  e.5)          send M to $j$;
  e.6)      $i$ exchanges reply message status with $j$;
  e.7)      perform steps f.1-f.7 for the reply messages from $j$;
For an ACK-vector(M,$j$) where $j \in CR_i$ or $j \in SB_i$:
        /* reply message for M arrives from a child node or a sibling node of $i$ */
  f.1)      if ACK-vector(M,$i$) $\supseteq$ ACK-vector(M,$j$), then ignore ACK-vector(M,$j$);
  f.2)      else begin
  f.3)          ACK-vector(M,$i$) $\leftarrow$ ACK-vector(M,$i$) or ACK-vector(M,$j$);
  f.4)          mark $j$ REPLIED for M;
  f.5)          if every $k \in CR_i$ is REPLIED,
  f.6)              call reply_ack;
  f.7)      end
For an ACK-vector(M,$j$) where $j \in PR_i$:
        /* reply message for M arrives from the parent node of $i$ */
  g.1)      if $E\text{-}SB_i$ is not initialized,
  g.2)          $E\text{-}SB_i \leftarrow SB_i \setminus \{k \mid \text{ACK-vector(M,}j\text{)}[k] = 1\}$;
  g.3)      ACK-vector(M,$i$) $\leftarrow$ ACK-vector(M,$i$) or ACK-vector(M,$j$);
  g.4)      if $E\text{-}SB_i \neq \varnothing$, then for every $k \in E\text{-}SB_i$,
   g.5)            send ACK-vector(M,$i$) to $k$;
   g.6)        for every $l \in CR_i$
   g.7)            send ACK-vector(M,$i$) to $l$;
For a $DES_j$ message;
   h.1)        for every $k \in SB_i$
   h.2)          if $k$ not $\in$ $E\text{-}SB_i$ and $k$ not $\in$ $DES_j$,
   h.3)            begin
   h.4)                $E\text{-}SB_i \leftarrow E\text{-}SB_i \cup \{k\}$;
   h.5)                send DCL-exsib($i$) to $k$;
   h.6)            end
   h.7)        for every $l \in CR_i$
   h.8)            send $DES_j$ to $l$;
For a DCL-exsib message from $j$;
   i.1)        $E\text{-}SB_i \leftarrow E\text{-}SB_i \cup j$;
   i.2)        send ACK-vector(M,$i$) to $j$;
For a FLUSH-OUT(M) message from $j$:
   j.1)        eliminate M from $BUF_i$; /* this message came along with a broadcast message */

Procedure reply_ack for $i$:
   k.1)        if $PR_i \in NEIB_i$, send ACK-vector(M,$i$) to $PR_i$,
   k.2)        else begin /* alternative-path search */
   k.3)                if $SB\text{-}DONE$ is not initialized,
   k.4)                    $SB\text{-}DONE \leftarrow \varnothing$;
   k.5)                if $E\text{-}SB_i \neq \varnothing$,
   k.6)                begin
   k.7)                    $SB\text{-}DONE \leftarrow SB\text{-}DONE \cup E\text{-}SB_i$;
   k.8)                    for every $k \in E\text{-}SB_i \setminus SB\text{-}DONE$
   k.9)                            send ACK-vector(M,$i$) and $SB\text{-}DONE$ to $k$;
   k.10)                   delete $SB\text{-}DONE$ from $i$;
   k.11)               end
   k.12)               for every $k \in CR_i$
   k.13)                       send ACK-vector(M,$i$) to $k$;
   k.14)           end
```

Some notes on the basic node algorithm: (a) In e.3 and e.6, when nodes exchange the status of messages, only the headers of the messages are exchanged so that the communication cost can be reduced. (b) A reply message can be rejected by the parent or a sibling if it contains no additional reply information, but a reply vector from a parent will not be rejected by a child (see algorithm g.*). Consider the following situation. During an alternative-path search, an external sibling v_e has received a reply vector from its sibling. v_e will forward the vector to its parent, assuming that additional reply information is contained in that vector. Suppose that a child node is allowed to reject a reply vector from its parent. If the vector is eventually routed back from the root node as a result of a link disconnection, v_e will reject this vector because it has already received the same vector. Consequently, some descendents of this rejecting node will not receive this reply vector. From Lemma 3.2, we know that the alternative-path may never be found. (c) For the source node S, if the vectors from its children indicate that all the nodes have received a particular broadcast message M (i.e., every bit is 1 in ACK-vector(M,S)), S attaches a FLUSH-OUT(M) to its next broadcast message.

3.4 Performance

3.4.1 Minimum Broadcast Delay

The broadcast delay of a message M for a single node i is defined to be the time delay from the time S starts to broadcast M to the first reception of M at node i, denoted by $D_i(M)$. The broadcast delay of M is defined to be $Max:\{D_j(M) | j \in$ V} Since it is assumed that "zero time" is spent at each node in assumption (3), $D_i(M)$ is the sum of the propagation delays of M on each communication link in a path from S to i and/or the time delays of waiting for the connections of some links. The delay caused by waiting for a link connection happens when the network is temporarily partitioned into several disconnected components.

Theorem 3.3 Minimum broadcast delay is achieved by the basic broadcast protocol.

Proof. For each node t, $t \neq S$, let $P = <S, v_1, v_2, \ldots, v_{i-1}, v_i, t>$ be the eventually connected path through which message M arrives at t using the basic broadcast protocol, and each node in P is the parent node of the next node (e.g., v_i is the parent node of t). We will show by contradiction that the delay for t to receive M is minimized if M travels along P. Suppose that P is not the shortest path to deliver M and $P' = <S, v'_1, \ldots, v'_j, t>$ is the shortest path. Let (v'_k, v'_{k+1}) be the first pair of nodes such that v'_k is not the parent node of v'_{k+1}, and that u is the parent node of

v'_{k+1}. Since the parent node is chosen as the first node from which M arrives, there must be a shorter path from S via u to v'_{k+1}, and then from v'_{k+1} to t. So, P' is not minimum, a contradiction. Hence, P is the minimum path from S to t for delivering M. □

Notice that, as each link disconnection results in an exchange of received messages on both sides of the link, partitioning of the network does not prevent messages from finding the shortest path to a node. Since the protocol deals with networks having arbitrarily changing topologies, it is possible that the order in which messages are received at a node is different from the order in which the messages were broadcast.

3.4.2 Communication Cost

The communication cost for a broadcast message M is defined to be the number of messages exchanged during the broadcast and reply phases of broadcasting M. Messages in the networks can be classified as broadcast messages and control messages; control messages consist of ACK-parents, ACK-siblings, ACK-vectors, and DCL-exsibs.

Lemma 3.5 The communication cost in the broadcast phase is $O(|E|)$.

Proof. Let M be a message broadcast from S. As the flood of M goes forward, each edge is traveled at most twice. The edge between a child and its parent is traveled only once by M because each node receiving the first copy of M from its parent will not send the same message back to its parent. The number of these parent-child edges in a network is $(|V|-1)$. So, in the worst case, the total number of the copies of M exchanged is $2|E| - (|V|-1)$. In the best case, each edge is traveled exactly once, which implies that $|E|$ copies of M are exchanged. The cost for control messages is $2|E|-|V|$. Therefore, the communication cost is $O(|E|)$. □

The analysis for the cost of control messages in reply phase is more complicated due to the fact that the network under which the protocol is performed is subject to link reconfigurations.

Lemma 3.6 The communication cost incurred by a link disconnection is $O(|E|)$.

Proof. The disconnection of a sibling link causes no message exchanges. A tree link (parent-child link) disconnection causes no problem if the child has a sibling. Otherwise, it causes a tree-splitting and invoke either an alternative-path search or a sibling reclassification. In the latter case, DCL-exsibs are sent to the "new" external siblings which were internal siblings before the link was broken. In the former case, ACK-vectors are sent to the tree's external siblings, which may result in subsequent alternative path searches on the other trees. In both cases, the total number of reply vectors exchanged over tree links is on the order of $O(|V|)$. It remains to be determined how many reply vectors are exchanged over the external sibling links which connect the affected trees. In the normal case, this number should be much smaller than $O(|V|)$. In the worst case, it should not exceed $O(|E|)$ (in a highly dense network). Hence, for each link disconnection, the communication cost incurred is $O(|E|)$.

Lemma 3.7 The communication cost incurred by a link connection is $O(|E|)$.

Proof. Each link connection will cause exchanges of broadcast status and reply vector status. If both sides contain the same reply information, no more messages are exchanged. If one side contains new reply information, the other side may invoke vector replying, which in turn may result in an alternative-path search. From Lemma 3.6, we know that the maximum cost induced is $O(|E|)$.

Lemma 3.8 The communication cost in the reply phase is bounded by $O(|V||E|)$.

Proof. In the best case, no parent-child links are disconnected during the reply phase; every reply vector can be routed to S via a primary path. Thus, the communication cost is $|V|-1$. Otherwise, there are 2 cases to be considered:

Case 1: communication cost incurred by link disconnections. During the reply phase, every link in the network is labeled either as a parent-child link or a sibling link. As shown in Lemma 3.6, a single link disconnection can cause at most $O(|E|)$ message exchanges. With the number of parent-child links bounded by $|V|-1$, the total communication cost induced by link disconnections can not exceed $O(|V||E|)$.

Case 2: communication cost incurred by link connections. We consider the following two cases. (i) Suppose that a link is connected within a tree. If the two nodes have identical reply vectors, it causes no additional cost except for the exchange of reply information. If two nodes have different reply vectors, both of them will send their newly acquired reply vectors to their parents. Since these two nodes are in the same tree, the reply vectors will stop at the first common ancestor of the two nodes. Hence, the cost of this connection is bounded by $O(|V|)$. (ii) Suppose that a link is connected between different trees. If both sides have different reply vectors, in the worst case it may cause the reply vectors be sent up and then down as a result of alternative-path search which is bounded by $O(|E|)$. Subsequent sibling link connections between the two trees will induce no cost in vector replies because the nodes in these two trees will have identical reply vectors by Lemma 3.3. Since there are at most $|V|$ different trees, the cost of link connections is at most $O(|V||E|)$.

Summarizing *Case* 1 and *Case* 2, it can be seen that the communication of the reply phase is bounded by $O(|V||E|)$. □.

Theorem 3.4 The communication cost of broadcasting a message is bounded by $O(|V||E|)$.

Proof . It follows immediately from Lemma 3.5, 3.6, 3.7 and 3.8. □

It is important to notice that, even at the extreme case where the network topologies are changing so rapidly that the number of link disconnections and connections is in the order of $O(|E|)$ (i.e., almost every link is reconfigured), the upper bound $O(|V||E|)$ still holds (i.e., not $O(|E|^2)$) because of the convergence of the reply information. From a practical point of view, in most of the mobile communication networks each node has only a fixed number of communication ports which are used to send and receive data. Therefore, the worst-case communication cost of the protocol is approximately $O(|V|^2)$.

3.4.3 Speedup in Reply Phase

In this section, we discuss a possible way of speeduping the reply phase. As each node tries to send the reply vector of a received broadcast message to S during the reply phase of the algorithm, the performance of the reply process depends on the number of link disconnections in the network. The number of link changes of a network in turn depends on the reconfiguration algorithms. For networks with good reconfiguration algorithms, very few links will be disconnected, which ensures that reply vectors can be quickly forwarded to a source node. Furthermore, the reply phase may be shorter than the broadcast phase if fewer links are disconnected during the reply phase because reply vectors are normally short packets compared to broadcast messages which may be arbitrarily long.

For a network that is undergoing rapid topological changes, Lemma 3.8 indicates that the reply phase may incur a substantial overhead in communication cost. This suggests that an improvement in the protocol is desirable, particularly in the speeding up of the reply phase.

A key observation to such an improvement is that, during the reply phase, the first node which detects an "all-1" reply vector may not be S. Recall that each node which receives the first copy of a broadcast message will put 1 into its own position in the ACK-reply (see line a.1 of the node algorithm). Thus, we can modify the algorithm by allowing any node (instead of only S) to issue a FLUSH-OUT message to remove a broadcast message M if it detects the fact that the message has been received by all the nodes. If node i is the first node detecting the fact, it discards M from its buffer memory and attaches a FLUSH-OUT(M) message to its own message to be broadcast. If i has no immediate messages of its own to broadcast, it can either pass the FLUSH-OUT(M) independently or pass it to its parent. Its parent will do likewise. When the FLUSH-OUT(M) reaches the root of a tree, it will be broadcast independently. This modification requires that an ACK-vector is appended to a broadcast message at the beginning during the broadcast phase. When the nodes on both sides of a link exchange a broadcast message (observe that they are candidate parents of each other), they also update their current ACK-vectors on that message.

It is difficult to analyze the performance gains by this speedup mechanism. However, in Figure 3.2 we show some network topologies in which significant speed-ups can be achieved by using the modified protocol. Notice that, in these examples, when t's start the reply phase, they immediately detect that all the nodes have received the broadcast message delivered along the flooding paths, so they can issue FLUSH-OUT messages and thus a high degree of speed-up is obtained. The degree of speed-up obtainable for a particular network depends not only on the network topology but also on the location of the source node; some nodes are better able to take advantage of the above modified algorithm than others.

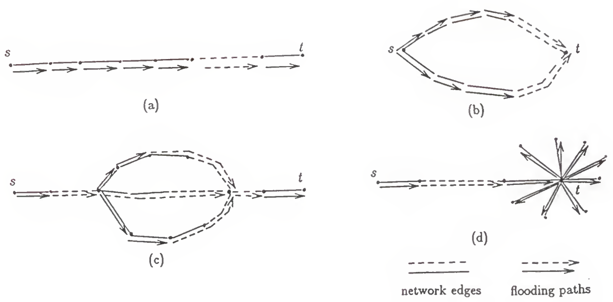

Figure 3.2 Some Examples of Speed-up

CHAPTER IV

GLOBAL NETWORK SURVEY

4.1 Background

The global survey problem (GSP) studied in this paper can be briefly specified as follows: Given a general network $G = (V,E)$ whose topological information is not available to any node in the network, and the network is vulnerable to unexpected link failures but remains connected, a global survey is conducted by a node $v \in V$; during the survey, a question Q, which allows a short answer (e.g., a message of constant length) respond, is broadcast to every one in the networks, how a distributed algorithm can be designed to complete the survey (the completion of a survey requires the answer from every node to be received in finite time) ? A simple instance of such survey that requires binary answers is *voting*; that is, all nodes in the network vote to decide a particular issue such as whether to allow a new node to join this network. Alternatively, we may consider the problem as that of global information collecting in the sense that a node in a dynamic network may occasionally wish to collect some specific information from every one. It is also closedly related to other research issues such as distributed consensus.

Designing a distributed algorithm for the GSP normally requires two essential operations: broadcast and reply. A survey question needs to be broadcast to all the nodes in the network before the answers to the question can be replied by the nodes. In this sense, the GSP can be considered as an extension of network broadcasting which has been extensively studied for more than a decade [3,10,11,14,25,33,35]. To the best of our knowledge, no previous work has addressed the GSP from the same perspective. A similar work was done by Baratz, Gopal and Segall [5] on fault-tolerant queries in networks. But the technique used in [5] is relatively straightforward and no optimization was attempted to reduce the message complexity, which is the primary concern of this work.

Like many other distributed computing problems, the performance measurement of an algorithm for the GSP involves time-message complexity trade-off. An efficient algorithm generally has higher message complexity while algorithms with low message complexities tend to take longer time to terminate. In this paper, we present lower bounds for the GSP on both time and message complexities. We show that $\Omega(|V|)$ is a time-complexity lower bound and $\Omega(\mathbf{min}(|V|,k)|V|+|E|)$ is a lower bound for message complexity. It is observed that the complexities on obtaining these two bounds are asymmetric. While there exists a straightforward algorithm to achieve a time complexity lower bounds, a message-optimal algorithm is complicated to design. Our main effort is to achieve the latter. Based on depth-first-search and multiple-tree traversal, we present a lower-bound algorithm in terms of its message complexity. The significance of the algorithm is that it achieves a message-complexity lower bound regardless of the number of link failures as long as the network remains connected. That is, even if the number of link failures during the survey is as large as $O(|E|)$, the message complexity of the algorithm is still bounded by $O(|V|^2)$.

4.2 The Model and Motivation

We consider this problem in networks in which the following things hold: (i) The network is subject to unpredictable topological changes. Consequently, it is assumed that for each node, the topological information is restricted to *1-neighborhood* (i.e., each node knows only to whom it is connected). This condition is the same as what has generally been assumed in other distributed computing problems such as broadcasts in dynamic networks [3,35], constructing MSTs and leader elections [4,16,29]. (ii) Each node has a unique id. which is available to every one; In this way, a node conducting a survey will be able to tell whether every reply has been received. It is also assumed that there exists a canonical order among these IDs. A straightforward implementation of such ordering would be ranking the nodes according to the lexical order of their IDs. (iii) The time to send a message over a communication link is arbitrary but finite. (iv) During a survey, some links may fail but nodes will not, which ensures that every node in the network will eventually reply. Despite the link failures, the network remains a connected component. A failed link is eliminated from the network and is not usable for the rest of the process. The rationale behind this is that, considering the communication delays in general computer networks, the time required to recover a typical link failure is unproportionally long.

The performance of a distributed GSP algorithm is measured according to two major criteria: time and communication complexities. The time complexity is the total time to finish a survey task and as commonly assumed, the exchange of a message over a link takes $O(1)$ time. There are basically two ways to define communication complexity. One is defined in terms of *bit complexity*. That is, the total network overhead is calculated according to the total number of bits transmitted over links in the network as a result of the algorithm. This definition is generally acceptable in many networks. However, from the perspective of a node processor, there are some overheads associated with each message transmitting, which are largely independent of the message length (e.g., channel allocation, routing decision, communication process generation etc.) Under these circumstances, the communication complexity of an algorithm may be defined in terms of its *message complexity*, which is the total number of message exchanges over the links in the network.

In some distributed computing problems, it is possible to bound the lengths of messages generated by an algorithm to a constant. Unfortunately, for GSP each node must reply with its ID. as well as its answer, otherwise the conducting node will not be able to tell which answer belong to which node and thus unable to determine whether a survey is complete. This implies that each message requires at least $O(\log|V|)$ bits.

As each answer replied to a question is short and of constant length, sending these messages on individual basis would flood the network. Therefore, the idea of *lumping* is considered a better alternative, which means concatenating many short messages into a slightly long but reasonable-length one. Through message lumping the message complexity of an algorithm can be significantly reduced. But a straight-forward lumping implies the possibility of creating messages of $O(|V|\log|V|)$-bit long. In this model, we manage to avoid the creation of such long messages and allow messages to be bounded by $O(|V|)$ bits. We shall prove that no algorithm based on message lumping can be designed without using messages less than $O(|V|)$ bits. In other words, $O(|V|)$ is a lower bound on message size. It is the main focus of this research to examine in detail the GSP algorithms based on lumping and their performances in terms of message complexity.

4.3 Lower Bounds

In this section, we proceed to show that $\Omega(|V|)$ and $\Omega(\mathbf{min}(|V|,k)|V|+|E|)$ are lower bounds on time complexity and message complexity for the GSP respectively. For the convenience of discussion, let v be the node that conducts the survey process for a question Q. That is, v is the source of the question Q and every node will have to reply to v with its answer to Q.

Theorem 4.2.1 Given k link failures during the survey process where k is arbitrarily large as long as the network remains connected, $\Omega(|V|)$ is a lower bound on time complexity for GSP.

Proof. Consider the case where the topology of a network is as shown in Fig. 4.1. The source node, v, broadcast Q to u_1, u_2, ... , $u_{|v|-1}$. Suppose that when these nodes receiving Q are ready to reply, (v, u_2), (v, u_2), ... ,$(v, u_{|V|-1})$ fails. All the replies must be forwarded via (v, u_1). Thus, given any protocol, it takes $O(|V|-1)$ time for v to receive all the replies. The theorem thus follows. □

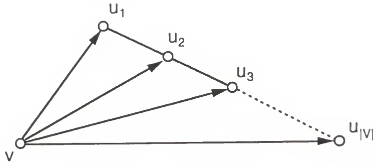

Fig. 4.1 An example for lower bound

It is interesting to observe that as the time complexity is computed with respect to the original network, an $\Omega(D)$ lower bound is not possible, where D is the diameter of the network. This is also shown in Fig. 1 where $D = 2$ but the time complexity is $O(|V|)$.

The survey for Q can be divided into two stages: broadcast stage and reply stage. As each node is assumed to know only its neighboring nodes, the only reliable way to broadcast the question Q to every node is message flooding. The flooding technique is very efficient and commonly used in sparse networks. Initially, v sends a copy of Q to all its neighbors. Each node, after receiving the first instance of message Q, keeps a copy and sends it to all neighboring nodes except the node/s from which Q arrived.

It is easy to verify that the flooding described above is a reliable algorithm to broadcast a message for a connected network. Thus a simple algorithm, the two-way flooding, can be constructed to obtain a time complexity lower bound for GSP (In [5] Baratz etc. use similar but slightly advanced idea to design algorithms for searching distributed resources.) This algorithm requires nodes to send their answers back to v by flooding.

Theorem 4.2.2 The two-way flooding algorithm is time-optimal for the GSP. Furthermore, it achieves minimum delay for the survey process.

Proof. As the flooding ensures that in the broadcast phase each node has its neighbors informed of the broadcast message Q, if the network remains connected in spite of link failures, for any node $u \neq v$, there must exist a simple path from v to u along which Q can be delivered to u. Since the network has $|V|$ nodes, the time for u to receive Q is at most $|V|-1$ units. Similarly, we can show that it takes at most $|V|-1$ units for v to receive the answer of Q from u if the reply is done by flooding. Therefore, the two-way flooding is time-optimal.

Now, we proceed to show that the algorithm achieves minimum delay for GSP. For each node $t \in V$, let $P = <v, v_1, v_2, \ldots, v_{i-1}, v_i>$ be the path through which Q arrives at t using the flooding, where $v_i = t$ and for all $1 \leq j \leq i$, v_j is the parent of v_{j+1}. It can be shown, by contradiction, that the delay for t to receive Q is minimized if Q travels along P. Suppose that $P' = <v, v'_1, \ldots, v'_{j-1}, v'_j>$ is a shorter path than P to deliver Q from v to t, where $v'_j = t$. Let k be the smallest index for a link (v'_k, v'_{k+1}) in P' such that v'_k is not the parent node of v'_{k+1}, and that w is the parent node of v'_{k+1}. Since the parent node is chosen as the first node from which Q arrives, there must be a shorter path from v to v'_{k+1} via w, and then from v'_{k+1} to t. which implies that P' is not minimum, a contradiction. Hence, P is the shortest path. Since a node sends its answer of Q also by flooding, we can see that the two-way flooding algorithm can achieve minimum delay. □.

Clearly, the cost of implementing the efficient two-way flooding is very high in terms of its message complexity, considering that the message complexity of the algorithm can be as high as $O(|V|^3)$ in edge-dense networks. In many circumstances, the existence of a large amount of messages in the network may result in message congestions, which will degrade the overall system performance. In this case, an algorithm with low message complexity would be more preferable. Nevertheless, the unpredictability of link failures significantly complicate the design of algorithms for such a purpose.

Definition 4.1 Let u be a node in G = (V, E) that received Q. The neighboring nodes of u can be classified into three categories: *parent*, *siblings*, and *children*. The parent of u is the node from which Q arrived first (ties are broken arbitrarily). Conversely, if the parent of u is w, then u is a *child node* of w. The neighboring nodes to which u is their parent form the children of u. The other neighboring nodes of u are its siblings.

Definition 4.2 The *p-neighborhood* of a node u is defined to be the set of nodes that can be reached from u by no more than *p-hops*, where each hop represents a crossing of a communication link. For each node u that has received Q, we denote the reply to Q by *R(u)* which contains at least the answer of u to Q. There exists at least one node in the network that is an *intended receiver* of $R(u)$. Conversely, if w is an intended receiver of u, then u is an *intended sender* of w.

Definition 4.3. A distributed algorithm for GSP is called a *p-neighborhood protocol*, if it satisfies the following axioms: For each node u, (1) its intended receivers are within its *p-neighborhood*; (2) if u is a receiver, it can either send its reply individually or send the concatenated message of its reply and the replies of its senders; (3) if all the replies from its intended senders have been received, it must reply in finite time; (4) any message arrives after u has replied to its intended receivers that contains new reply information should be forwarded by u in finite time.

Axiom (4) implies that if a node is disconnected from its intended receiver, it will be forced to forward its reply to one of its non-intended neighbors. This neighbor, receiving an unintended message, must forward it independently as its lumping stage has ended.

It can be observed that p-neighborhood protocols can be most realistically implemented when $p = 1$. The reason is simple: when $p \geq 2$, a message intended for a non-neighboring node should be routed through intermediate nodes. Nevertheless, as each node has only local topological information, to reliably deliver messages to the intended receivers, *local-flooding* (i.e., flooding messages within p-neighborhood), may be necessary. This will inevitably result in high message complexity. One alternative is for each node to construct routing paths through local information exchanges within its p-neighborhood, which again may also require *local-flooding*. Furthermore, even if these routing paths can be established, link failures will certainly complicate such routings. Consequently, we are most interested in the protocols of 1-neighborhood lumping. In the next theorem, we show a message complexity lower bound of p-neighborhood protocols for GSP in networks with unpredictable link failures.

Theorem 4.2.3 Given k link failures during the survey process where k is arbitrarily large but the network remains connected, $\Omega(\mathbf{min}(|V|,k)|V|+|E|)$ is a lower bound on message complexity of 1-neighborhood protocols for GSP.

Proof. As flooding is considered the only reliable broadcasting protocol during the broadcast stage, the message complexity for broadcasting Q to all the nodes in the network is bounded by $O(|E|)$. From the axioms of the 1-neighborhood protocols, each node has at least one intended receiver in its adjacent nodes to which a reply of Q can be sent. It is easy to show that if a node chooses a child as its intended receiver, it is possible that its reply will never be forwarded to the source (e.g., consider the case that the edge connecting the node and its child is a *bridge* of the network). Therefore, it is clear that for each node, its parent and siblings are the only possible candidates for intended receivers as they are the nodes from which Q arrived. That is, despite no global topological information is available to any one, the flooding of Q implies that there exists at least one path from the parent or a sibling to the source. Thus, in the reply phase, these properties give us the following possible cases to consider.

Case 1: (*no lumping*) a node sends out its answer without waiting for its senders' replies. In Fig. 4.2, we can see that it takes $(|V|-1)$ messages to broadcast Q and in the worst case there are $1 + 2 + \ldots + (|V|-1)$ messages needed to reply the answers. Hence, the number of messages exchanged over links is $(|V|-1) + 1 + 2 + \ldots + (|V|-1)$ $= (|V|-1)(|V|+2)/2 = O(|V|^2)$. This situation occurs even without a single link failure.

Case 2: (*lumping*) we consider the case where each node may reply after lumping together the replies from its intended senders.

Case 2.1: a node chooses its parent as its intended receiver. Consider the following case shown in Fig. 4.3, $v_{|V|}$ is the common parent of u_1, u_2, ... , and u_k. Suppose that all the parent links of u_2, ... , u_k fail before the nodes are ready to reply. Clearly, $v_{|V|}$ will start replying to its parent when it receives the reply from u_1 as u_1 is its remaining connected child now. The replies from u_2, ... , u_k have to be delivered via u_1 as it is the only node leading to v. According to the Axioms, in the worst case where u_2 replies before the answer from u_3 arrives, u_3 replies before it receives u_4's answer, and so on, it causes $O(k|V|)$ messages exchanged.

Case 2.2: a node chooses its siblings as its intended receivers. Consider the example shown in Fig. 4.4. As w_i and u_{k-i+1} are siblings for all $1 \le i \le k$, these siblings will exchange their replies. However, all the paths leading to v must go through v_1, v_2, ... , $v_{|V|}$. Similar to Case 2.1, in the worst case the number of message exchanges can be $O(k|V|)$.

Case 2.3: a node chooses its parent and siblings as its intended receivers. Consider the case shown in Fig. 4.5. u_{2i} and $u4i$ will collect the reply from $u3i$ for all $1 \le i \le k$ before they reply; u_2 will collect all the replies from $u_{21}, u_{22}, \dots , u_{2k}$ before it replies; and u_4 will lump together the replies from u_{4i} for all $1 \le i \le k$. Suppose that the links between u_{2i} and u_{3i} are disconnected before the replies from u_i's can be delivered, they must be routed via u_4. Since $u_{21}, \dots , u_{2k}$ are not in u_4's 1-neighborhood, The worst case is that their replies are routed individually through u_4, which results in $O(k|V|)$ messages being delivered. □

As we have point out the difficulty of implementing an algorithm adopting message lumping with neighborhood ≥ 2, nevertheless, we can extend the previous theorem to show that widening neighborhood in general does not reduce message complexity.

Theorem 4.2.4 Given that k links fail during the survey process where k is arbitrarily large but the network remains connected, $\Omega(\mathbf{min}(|V|,k)|V|+|E|)$ is a lower bound on message complexity of p-neighborhood protocols for GSP with p being an arbitrarily large constant.

Proof . The proof can be extended directly from that of Theorem 4.2.3 by the technique of "node padding". Detailed proof is omitted here. The basic idea is to insert sufficient number of nodes to push some critical nodes beyond some neighborhoods. More specifically, consider the example in Fig. 4.5. By inserting p colums of nodes between u_{3i} and u_{2i} where $1 \le i \le k$, the lumpings of u_4 does not include replies from $u_{21}, \dots , u_{2k}$ since they are beyond u_4 p-neighborhood. This can force these messages being replied individually through u_4 in case that the links between u_{2i}, for all $1 \le i \le k$, and u_2 are disconnected. The arguments of other cases can be similarly constructed. □

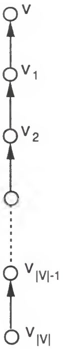

Fig. 4.2 Case 1

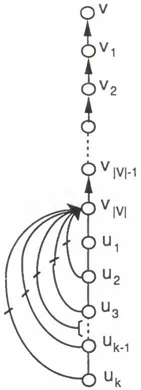

Fig. 4.3 Case 2.1

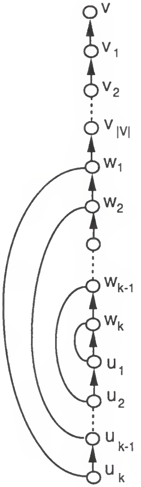

Fig. 4.4 Case 2.2

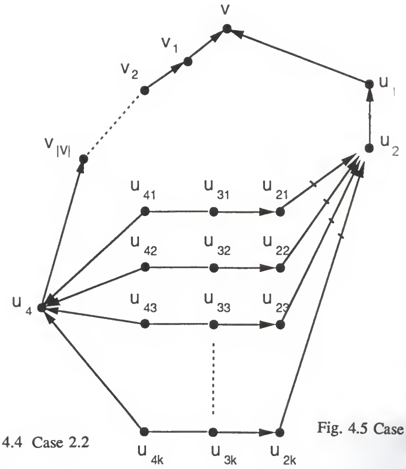

Fig. 4.5 Case 2.3

4.4 Optimizing Message Complexity

The observation that enlarging a node's lumping neighborhood does not yield a better lower-bound result leads us to focus on exploring better routing strategies. Without a global picture of the network, each node is forced to make routing decision on local basis. The routing decisions, considered in a broad sense, include not only message forwarding but also selecting a neighborhood, choosing intended receivers, etc.

Our main effort, in this section, is to present an algorithm which is reliable and optimal in terms of message complexity for arbitrarily large number of link failures. The underlined strategy of this algorithm is based on the concept of *alternative-path search.* The algorithm belongs to 1-neighborhood protocol. Within its 1-neighborhood, each node has a unique intended receiver for its reply message that is its parent node. In the operational sense, the algorithm can be roughly divided into two phases: broadcast phase and reply phase. In the temporal sense, there is no clear distinction; some nodes may be still in their broadcast phases while the others may be already in their reply phases.

In the broadcast phase, each node performs the flooding procedure [33][35]. The message Q broadcast from v initially contains only the question. As this message is disseminated across the network, each node receiving an incoming Q will do the following things: (1) keep a copy, (2) send it to all the neighboring nodes except those from which Q arrived. According to Definition 4.1, each node establish a relationship with each of its neighbors. Considering only the parent-child relationship among nodes in the network, a spanning tree is formed with v, the source of Q, being the root of the tree. The path connecting a node u to v that consists of only tree edges is the *primary path* of u. The primary path of u is a shortest path along which Q arrives. It may not be the unique shortest path from v to u. Under the assumption of unpredictable propagation delay of a link, it is not unlikely that the primary path of u may change in the next survey. Nevertheless, a primary path does provide a useful hint for replying answers (i.e., with limited topological information, it is reasonable to choose the primary path to send answers back to v in the reply phase).

The reply phase is initiated by nodes which satisfy one of the following conditions: (1) it has no neighbors to which Q needs to be sent or (2) all the links to its children are disconnected. In both cases the nodes are called *leaves*. A leaf node will send its answer to its parent immediately. If a node is not a leaf node, it replies with a message that is the concatenation of its own reply to Q and the replies from its children. The message is then forwarded to its parent.

Link failures during broadcasting are handled by discarding the link. Failures that occur during the reply phase have different implications. Let l be a communication link that fails. There are two cases to be considered. If l is a sibling link, no action is neeed to be taken. If l is a tree link, proper algorithms are required to ensure the reliability of the protocol, which will be discussed in the next two sections.

Definition 4.4 Let v_h and v_l be the nodes connected by tree link l and v_h is the parent of v_l. The failure of l makes v_l become the root of a new tree, denoted by T_l. This new tree T_l is said to be a *low tree*. In contrast, the "old tree" (i.e., the tree that contains v_h) is called a *high tree*, denoted by T_h.

The birth of a new tree has significant consequences for routing reply messages back to the source v. For instance, an alternate-path search is needed if v_l has not yet replied with its answer message, because the primary path connecting v_l and v no long exists.

Definition 4.5 Two nodes are said to be *internal* siblings to each other if they are in the same tree and connected by a non-tree link; They are *external* siblings if they are not in the same tree but are directly connected.

4.4.1 The Message Structure

Before we get into the detail of our protocol, it is necessary to discuss the way each message is constructed. The structure of a broadcast message is quite simple; it contains the question and the source. Thus it takes $O(\log|V|)$ bits. A reply message requires more complex design. The reason is that it will contain not only answers to the question but also some routing information for maintaining good routing properties.

The reply message structure contains mainly four parts: i) answers to the survey problem; ii) *reply-record* which consists of the nodes that provide these answers; iii) *descendent-record* which, created by the root of a tree, is the set of node within the tree; iv) *visited-node-record* which is the set of nodes that have been visited; v) *visited-tree-record* consisting of trees, presented as a set of nodes, that have been partially visited, i.e., some of the nodes in the trees may yet to be explored. The purpose of reply-record is to keep track of the nodes that completed their replies. A descendent-record is used to determine whether a sibling is internal or external. The purposes of visited-node-record and visited-tree-record will be apparent in the coming sections.

Except for the answers, which are of constant length each , all the three records are represented as bit-vectors. Each node is associated with a position in the records. These positions are arranged in a canonical order so that each node knows which position belongs to which node. A bit 1 is set for a node in its corresponding position at the reply-record if its answer is contained in the message; otherwise it remains 0. In order to enforce a one-to-one correspondence between answers and reply-record, the answers are arranged in the same order as the reply-record. For example, the n th answer is associated with the n th bit that has been set to 1. For descendent-record bit 1 represents "is-in". Similarly, for the rest of two records, bit 1 denotes "visited". The size of each reply message structured in this fashion is $O(|V|)$ bits.

4.4.2 Alternative Path Search

If none of the failures occurs at tree links, all the replies are ensured to be forwarded to v via tree links. However, this is not always the case. A tree link failure imposes great routing difficulty on parent as well as child. In the next two sections, we show how a failure is properly handled if the failed link is a tree edge. Two different processes will be invoked when a new tree T_l is splitting from an old tree T_h. Each process is initiated by one of the two nodes connected via the tree link whose failure results in a tree splitting. Let us call the algorithm executed by the child a *low-tree algorithm* and the one undertaken by the parent the *high-tree algorithm*.

4.4.2.1 Low-tree algorithm

The low-tree algorithm is the core of the alternative path search. The main task of this algorithm is to route messages from nodes in the low tree T_l to v. Since the link between v_l and v_h failed, messages from T_l to v must be routed via non-tree links. One way to achieve this is by flooding at v_l, which is similar to the algorithm proposed by Baratz, Gopal and Segall [5]. The advantage is its simplicity, but its message complexity is not optimal. Instead of flooding at v_l, we provide an alternative-path-search algorithm based on *depth-first-search* and *loop-free routing*. The algorithm is initiated at v_l.

In this algorithm, v_l will try to find a sibling first. If it succeeds, the reply from v_l is forwarded to the sibling. Otherwise, the message is sent back to one of its children. A message that is sent from a parent to a child is called a *backtracked message*.

A child node receiving a backtracked message from its parent attempts to send this message out of the low tree by identifying an external sibling. This is crucial to the low-tree algorithm in terms of optimizing the message complexity since passing a message to an internal sibling does not further the goal of forwarding the backtracked message to the source as the message will remain in the same tree.

There is a static way of discerning external siblings from internal siblings. From Definition 4.1, we can see that two siblings are internal if their *least-common-ancestor* is in the same tree as they are, otherwise they are external siblings. Consider an algorithm as follows: During the broadcast phase, each node broadcasts Q that is appended with its node id. By exchanging its broadcast message with a neighboring node, a node can identify whether the neighbor is an external sibling by checking the *ancestor-record* of the neighbor. The advantage is that it can be done off-line. Nevertheless, implementing such procedure may generate, in the worst case, broadcast messages that are as long as $O(|V|\log|V|)$ bits due to the concatenation of node IDs.

In our algorithm, external siblings are identified dynamically. The method is to allow the root of a tree to pass the necessary information in the form of a descendent record which contains all the node IDs. of the tree. A node can extract the descendent-record from the backtracked message it received and then the decision is straightforward since a sibling not in the record must be external. The low-tree algorithm consists of three major components which interact with each other.

A. The depth-first-search. In this procedure, the backtracked message is routed according to the following precedence: external siblings, child nodes, and the parent. That is, a node receiving a backtracked message will always try to find an external sibling. If it succeeds, the message is forwarded to the sibling. If the message is eventually returned by the sibling (i.e., the sibling fails to find an alternative path to v) or the link connecting it to the sibling fails (thus, it is not certain if the routing can succeeds via this sibling), another external sibling is tried. If it has no external sibling or all the external siblings have been tried and no alternative path is found, the backtracked message is sent to one of its children if it has any; if it does not have any child node, the message is sent back to its parent. If that child fails its search and returns the message, it tries another child until all the children have been explored. In this case, the message is sent back to its parent. By doing this, either the message will eventually be sent out of this low tree or all the external siblings needed to be explored have been explored. If it is the latter case, the backtracked message is returned to the external sibling from which it was delivered to this low tree.

It is worth mentioning that in order to avoid the repetition of fruitless sibling exploration, each external sibling that has returned a backtracked message is *deactivated* and will not being explored for the routing of subsequent messages unless it is reactivated.

B. Loop-free routing and new-tree broadcasting. It is clear that during the search of an alternative path, an external sibling should not be visited by the same backtracked message more than once. That is, the path that a backtracked message traverses should be loop-free. To achieve this, each reply message is associated with a visited-node-record. Each time a reply message is received by a node, the id. of the node is stamped to the visited-node-record. Thus, an external sibling visited before by the message can be excluded from being explored.

In addition, we see some practical reasons for the algorithm to keep loop-free the routing of a backtracked message in the "tree level" as well as in the node level. In other words, a message is not to be sent to those trees which have been visited by the same message earlier even though the trees contain nodes that have not been visited by the message.

There are two major reasons for loop-free traversal in the tree level: First, for reducing the number of message exchanges as a non-loop-free traversal will inevitably cause more nodes to be visited. Second, for good management of routing paths; a non-loop-free traversal is difficult to manage as it may cause problems when messages start backtracking, especially when a traversal path is disconnected by link failures. For such a purpose, a visited-tree-record is also required for each reply message. When a root of a tree receives a message, it stamps the visited-tree-record with its tree information. Therefore, an external sibling in the visited-tree-record is avoided from being explored by the message. There is, however, some inherent complexity in maintaining a loop-free traversal in the alternate-path search on dynamic and faulty networks. The following example illustrates this.

Example 1. Consider that tree T_1 receives a message M_a from one of its external siblings in its neighboring tree, denoted by n_e, and that T_1 consists of two parts, $T_{1,1}$ and $T_{1,2}$, connected by l which is the only link connecting $T_{1,1}$ and $T_{1,2}$. An alternative-path search proceeds in such a way that message M_a visits trees T_2, T_3, ... ,T_{j-1}, and T_j, one after another as shown in Fig. 4.7. Now, suppose that T_j has no external siblings other than those in $T_{1,2}$. Since the traversal of the backtracked message M_a should be loop-free at tree level, it will not be sent to $T_{1,2}$. Therefore, M_a backtracks to T_{j-1}, T_{j-2} and so on. In the meantime, link l fails, which results in the split of $T_{1,2}$ from T_1 as a new tree. Consequently, when M_a is eventually returned to T_1, $T_{1,1}$ will return M_a to n_e. Hence, $T_{1,2}$ is not explored by the alternative path search. If $T_{1,2}$ happens to be the only tree via which M_a can be sent to v, the search will fail to find the only path leading to v. This makes the protocol unreliable. Therefore, the situation must be avoided.

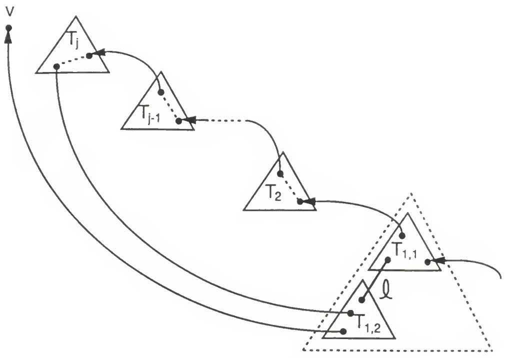

Fig. 4.7 An example

A reasonable way to prevent this is as follows: Once a new tree is born (split from the old tree), this information is broadcast to all of its neighboring trees. This ensures that if these neighboring trees have any backtracked messages to forward, the messages can be sent to this new tree. If a backtracked message can be delivered to v via the newly-born tree, a *reactivation* message is created. A reactivation message is used to inform those external siblings that the external siblings in the newly-born tree can be re-explored for further routings. A reactivation message is sent along the path via which a message backtracked. When a node receives a reactivation message from a deactivated external sibling, it will reactivate this sibling.

The algorithm to broadcast a *tree-born* message from the newly-split tree is designed by the same principle of the alternative path search of a backtracked message. That is, the tree-born message initiated by the root traverses in the tree in a depth-first-search manner. When a child receives the message from its parent, it sends a copy of the message to all of its external siblings which have not yet received the message. To achieve this, each time a message is sent to an external sibling, the id.

of the sibling is kept in the visited-node-record of the message, which is similar to the alternative path search of a backtracked message. By doing so, the external sibling can be exempted from receiving the same tree-born message more than once from nodes in the new tree. On the other hand, once a node receives the message of a new tree information, the message is forward to its root. After being informed of this information, the root will broadcast this message to all its descendents via tree links.

C. Sibling reclassification. The third part of the low-tree algorithm is to maintain the correct relationship between siblings. When the low tree is disconnected from the high tree, the nodes in the high tree that are the siblings of the low tree must be "re-classified" as external siblings since they are no long in the same tree. The root of the low tree must broadcast a *reclassification* message to its descendents. This is done by sending such message, which contains the IDs. of the descendents, along the tree links. Each node receiving the message will redefine its sibling relationship accordingly. Those "newly-born" external siblings of the low tree will subsequently be explored by the alternative path search. Note that there is no need to send a message across the links between the low tree and the high tree as the reclassification action will be taken by the high tree as well.

4.4.2.2 High-tree algorithm

We now consider the algorithm for v_h in the case where v_h loses one of its children because of link failure. Tasks need to be handled are included in the following procedures:

A. Resumption of alternative-path search. If v_h is currently conducting an alternative-path search on this disconnected branch, v_h will abort the search on the failed branch and resume the search on the other child branches. This may result in two alternative-path search processes concurrently exist, one in the low tree and another in the high tree. The necessity of such action is clear since v_h is unable to know whether the disconnected search in the low tree will succeed or not.

B. Sibling reclassification. Similar steps need to be taken by the high tree to readjust the status of its siblings. This is done by sending a message (i.e., a descendent-record which consists of nodes in the low tree) from v_h along tree edges to the root of the high tree. Once the root of the high tree receives such a message, it will broadcast a reclassification message, also containing nodes in the low tree, along the tree links. A node receiving a reclassification message will consider those siblings in the low tree external.

C. Exploration of new siblings. As v_h is not the root of T_h but a leaf of the tree, it is possible that when the root of T_h receives the message sent by v_h, which indicates a link failure, an alternative-path search (invoked by the root of T_h as part of its low-tree algorithm) may be already undergoing in T_h. If this is the case, these new external siblings, previously excluded because they were internal siblings, should be explored for routing backtracked messages in T_h. It is not unlikely that pathes from the high tree to v may go through these new external siblings of T_h.

Notes: i) During the reply phase and in both of the low-tree and the high-tree algorithms, a message from a child or a sibling may arrive that contains no additional reply information. This message will be rejected. ii) Each backtracked message has its own alternative-path search. Thus, it is not unlikely that backtracked messages in two adjacent trees may explore each other. Consequently, there may exist more than one backtracked message in a tree. In this case, each backtracked message, which can collect as many reply messages along its alternative-path search by merging the bit-vectors of the reply-records encountered into its own, will proceed independently. iii) Also note that the low-tree algorithm and high-tree algorithm are the necessary steps to be performed from a low tree and high tree point of views. These algorithms are separately discussed mainly for the convenience of illustrating the whole protocol. In fact, it is possible that in a tree, both algorithms may be invoked concurrently by different nodes. In this case, these algorithms will proceed independently in the tree.

4.4.3 Validation of the Protocol

Now, we shall show that the protocol is reliable. The basic idea of the proof is to show that no external siblings ever needed to be explored are not visited during the alternative-path search. That is to show that the alternative-path search of the low-tree algorithm is exhaustive.

Lemma 4.3.1 If each tree link is not disconnected before the child node finishes replying, then every reply message can be sent to v through the primary path.

Proof. If a tree link is not disconnected before the child node finishes replying its vector, then each node can reply to its parent via the tree link. It immediately follows that each reply message will be forwarded to v through a path consisting of only tree links, which is the primary path. □

Lemma 4.3.2 The alternative-path search is exhaustive.

Proof. Let T_i be a tree that has a backtracked message M_b to send out of this tree. Observe that there are three groups of external siblings of T_i that will not be explored by M_b when an alternative-path search is conducted in T_i.

The first group of external siblings are those contained in the visited-node-record of M_b since they are already been explored.

The second group of external siblings are those contained in the visited-tree-record of B but not in the visited-node-record of B. As discussed in the low-tree algorithm, this is to ensure that a loop-free traversal for a backtracked message can be maintained. However, these external siblings will be explored in either of the two conditions: 1) M_b is eventually returned to the trees that contain such external siblings since these siblings have not yet been explored by these trees (they are not in the visited-node-record of M_b). 2) The trees that contain these external siblings separate from their original trees due to link failures. This will force the newly-born tree to broadcast a tree-born message to T_i, which will cause T_i to send M_b to these external siblings. Therefore, eventually these external siblings will be explored by M_b if they ever need to be.

The last group of external siblings are those "originally internal" siblings of T_i. These siblings are contained in a tree splitting from T_i and thus become external siblings of T_i. And these siblings will be explored by M_b once a *reclassification* message arrives at the root of T_i as illustrated in the high-tree algorithm.

From the above cases discussed, it is clear that no external siblings are ignored by the alternative-path search if they are in the paths leading to the source v. □

Theorem 4.3.1 The protocol guarantees that every reply message will be delivered to v.

Proof. Let message M_a be the answer of node u to the question Q. If no link in the primary path from u to v fails, from Lemma 4.3.1, we know that M_a can be sent to v.

Suppose that the primary path from u to v is disconnected during the reply phase. Let v_1 be the first node encountered by M_a in the primary path such that v_1 is disconnected from its parent. According to the low-tree algorithm, v_1 is the root of a tree T_1 and an alternative-path search will be initiated when M_a arrives at v_1. From Lemma 4.3.2, we know that M_a will exhaustively explore all external siblings in the neighboring trees that are needed to be explored. Since T_1 is not isolated from the rest of the network, there must exist a tree, say T_2 rooted at v_2 such that M_a can be delivered to T_2. If v_2 is not v, another alternative-path search process will be initiated upon the arrival of M_a at v_2. Consequently, the reply of the message M_a may incur a number of alternative-path search processes in trees.

We show, by contradiction, that the alternative-path search of M_a will eventually succeed. Suppose that there exists a path from v_1 to v and the alternative-path search of M_a terminates without finding it. Since the network remains connected despite of link failures, there must exist two adjacent trees, T_i and T_j, such that T_i has a path to v but unexplored by M_a and that T_j is explored by M_a. The fact that T_i is not explored by M_a from T_j gives us two possible cases to discuss:

Case 1: T_i contains nodes which are in the visited-tree-record but not in the visited-node-record of M_a. This implies that T_i must be a newly-born tree which is split from the tree that have been explored (partially) by M_a, otherwise it will be explored. But as shown in the low-tree algorithm, a newly-born tree will broadcast a message to its neighboring trees and thus invite a backtracked message. This means that T_i will invite a M_a from T_j, a contradiction.

Case 2: T_i contains nodes neither in the visited-node-record nor in the visited-tree-record of M_a, which implies that T_i is a new tree split from T_j. In response to that, both the low-tree algorithm and the high-tree algorithm will do sibling-reclassification, which ensures that after finite time, siblings between T_i and T_j will be classified as external. Therefore, M_a must be sent to T_i; again, this is a contradiction.

Therefore, we conclude that if the network is connected, every reply message will be routed to v. □

From Theorem 4.3.1, the following theorem immediately follows.

Corollary 4.3.1 The protocol is reliable if no link failure causes partitioning of the network. □

4.4.4 Computational complexity

In this section, we discuss the computational complexity of the algorithm. The algorithm is designed to achieve the message complexity lower bound for GSP.

Theorem 4.4.1 The message complexity of the algorithm for GSP is $(|V|^2)$ with arbitrary link failures as long as the network remains connected.

Proof. Without loss of generality, assume that all of the link failures occur in the reply phase. As the survey process proceeds, each tree link failure will increment the number of trees by one. Let $T_1, T_2, \ldots, T_{n_t}$ be all the trees in the network at the end of the process with $n_t \le |V|$.

Consider a tree T_i where $1 \le i \le n_t$. Let M_a be a reply message that arrives at T_i from a neighboring tree T_j (a backtracked message). Let us count the maximum number of message exchanges in T_i for forwarding M_a to v. The message exchanges occur either at tree links or external sibling links.

First, we consider how many messages are possibly exchanged over tree links due to the routing of M_a to v. The worst case occurs when the low-tree algorithm can not deliver M_a to v via any of its external siblings so that M_a is eventually returned to T_j. Since the alternative-path search works in a depth-first-search manner, the number of messages exchanged in T_i caused by the arrival of M_a is bounded by $O(|T_i|)$. There are at most $|V|-|T_i|-1$ such backtracked messages like M_a that may reach T_i, excluding the source v (recall that a duplicate message from a sibling or a child will be rejected). Hence, during the whole process, the total number of messages exchanged over tree links in T_i is at most $O(|T_i|(|V|-|T_i|-1))$.

Now, we compute the upper bound on the number of messages exchanged over the external sibling links of T_i. Suppose that there are $e_{neig,i}$ external sibling links for T_i. For each message M_a, each time when M_a is returned from an external sibling or when the chosen external sibling link fails, the algorithm has to find another external sibling to retry. This increments the number of messages exchanged over sibling links by one. The alternative-path search ensures that if a reply message is returned from a neighboring tree, the tree will not be explored again unless the external siblings in the tree are reactivated. Let $r_{neig,i}$ be the total number of external siblings of T_i that have been reactived. This makes the total number of "retries" at T_i less than $e_{neig,i}+r_{neig,i}$. Again, no more than $|V|-|T_i|-1$ such backtracked messages like M_a may arrive at T_i. Therefore, during the survey process, the total number of messages exchanged over the sibling links connecting T_i and its neighboring trees are bounded by $O(e_{neig,i}+r_{neig,i}+|V|-|T_i|-1)$.

Summing up the two terms for all $T_1, \ldots, T_{n_t}$ gives the following total number of messages exchanged during the reply phase,

$$\sum_{i=1}^{n_t} (O(|T_i|(|V|-|T_i|-1)) + O(e_{neig,i}+r_{neig,i}+|V|-|T_i|-1))$$

$$= O(|T_1|(|V|-|T_1|-1)) + O(e_{neig,1}+|V|-|T_1|-1) + O(|T_2|(|V|-|T_2|-1)) + O(e_{neig,2}+|V|-|T_2|-1)$$
$$+ \ldots + O(|T_{n_t}|(|V|-|T_{n_t}|-1)) + O(e_{neig,n_t}+|V|-|T_{n_t}|-1) + O(r_{neig,1} + \ldots + r_{neig,n_t})$$

$$= \{O(|T_1|(|V|-|T_1|-1)) + O(|T_2|(|V|-|T_2|-1)) + \ldots + O(|T_{n_t}|(|V|-|T_{n_t}|-1))\}$$
$$+ \{O(e_{neig,1}+|V|-|T_1|-1) + O(e_{neig,2}+|V|-|T_2|-1) + \ldots + O(e_{neig,n_t}+|V|-|T_{n_t}|-1)\}$$
$$+ O(r_{neig,1} + \ldots + r_{neig,n_t})$$

$$\leq O(|V|(|T_1|+|T_2| + \ldots + |T_{n_t}|)) + O(e_{neig,1}+e_{neig,2} + \ldots + e_{neig,n_t}) + O(n_t|V|)$$

$$+ O(r_{neig,1} + \dots + r_{neig,n_t})$$

$$\leq O(|V|^2) + O(|E|) + O(|V|^2) + O(r_{neig,1} + \dots + r_{neig,n_t})$$

$$= O(|V|^2) + O(r_{neig,1} + \dots + r_{neig,n_t})$$

Since each time a new tree is generated, at most $O(|V|)$ external siblings in the network can be reactived, $O(r_{neig,1} + \dots + r_{neig,n_t}) < O(n_t|V|) < O(|V|^2)$. So, $O(|V|^2)+O(r_{neig,1} + \dots + r_{neig,n_t}) = O(|V|^2)$.

Finally, let us count the number of message exchanges due to maintenance purposes, i.e., the sibling reclassification and tree-born message broadcast. It is clear that sibling reclassification involves only tree edges traversals; thus the total number of message exchanges due to this factor is bounded by $O(|V|^2)$. Similarly, tree-born message broadcast is done in depth-first-search fashion and involves tree-traversals and message exchanges over external sibling links that is bounded by the number of neighboring trees. Therefore, it incurs no more than $O(|V|^2)$ message exchanges. Hence, including the $O(|E|)$ messages exchanged during the broadcast phase and the $O(|V|^2)$ reactivation messages exchanged as a result of the generation of new trees, we conclude that the algorithm has an upper bound of $O(|V|^2)$ as its worst-case message complexity. □

Corollary 4.4.1 The algorithm for GSP has optimal message complexity. □

An interesting open question is whether there exists a protocol for this global survey problem which can achieve the lower bounds on both time and message complexities.

CHAPTER V
CONCLUSION

In this closing chapter, we summarize what have been achieved during this research and suggest some open problems for future research.

5.1 Major Results

In Chapter 2, We present a hierarchical, partitioning approach to the network reconfiguration problem of large-scale satellite networks. By partitioning a satellite network space into regions of approximately equal sizes, the reconfiguration problem of the network can be reduced to a number of regional link assignment problems which, due to their relative simplicity, can be solved recursively by connecting satellites in a layer-by-layer manner. The algorithm proves to be superior to other methods in terms of time complexity.

It is also shown that, in the large-scale networks where each region contains at least two nodes, the node-connectivity of the network is maximized when each node is restricted to have four transceivers. With this property, packet routing is more flexible, and the possibility of packet congestion can be effectively reduced. In addition, we also provide some fault-tolerant mechanisms which enable us to dynamically handle the unexpected topological changes such as link failures or node failures.

In short, our effort on studying satellite network reconfiguration has resulted in a significant method. Unlike the NpNs model, the method provides a more general network model which does not impose any restriction on the orbits of the satellites. Consequently, it is free of the vulnerability that the NpNs model suffers.

In Chapter 3, we have discussed the issues of message broadcast in networks which are subject to frequent link reconfigurations but are eventually connected. The protocol presented has been shown to achieve reliability, efficiency, and finite message buffering (previously only infinite-buffered algorithms are available). We believe that the protocol has significant implications for broadcast protocol designs in communication networks with such characteristics, in particular, packet radio networks and point-to-point satellite networks.

With minor modifications, the protocol can be extended to serve as an information-collecting algorithm in eventually-connected networks. Information collecting is necessary in networks where a node may need specific information from every node in the network. For example, the network may need to select a master file server or elect a leader so that some administrative things can be centralized. Hence, the reply phase of the algorithm can be modified to bring back not only the reply vector but also the needed information from each node. The algorithm also can be applied to other issues such as network votings and synchronizations.

Based on similar but relatively restrictive model, we discussed another research topic in Chapter 4: network survey (or GSP, Global Survey Problem). The problem assumes that the network suffers link failures which may not be recovered during the survey process and that the network remains connected despite of these failures. We show that $\Omega(|V|)$ is a lower bound on time complexity and a message-complexity lower bound is $\Omega(\mathbf{min}(|V|,k)|V|+|E|)$ for lumping algorithms.

Previously some works have been done on fault-tolerant network resource allocation in which a modified flooding algorithm is provided. The algorithm can be converted to an algorithm for this problem in a straightforward manner. However, the algorithm has $O(|V||E|)$ message complexity, which is not optimal, in particular when $O(|E|) = O(|V|^2)$. In this research we design a loop-free algorithm, which consists of the lower-tree algorithm and the high-tree algorithm. The algorithm is bounded by $O(|V|^2)$ regardless of the number of link failures, thus an optimal algorithm in terms of message complexity.

5.2 Future Research Directions

Future research on network reconfiguration includes extending the hierarchical partioning approach to handle more general cases (e.g. cases in which the number of links in a node is not fixed). Another interesting direction for future work is to design the hierarchical routing algorithms which can take advantage of the hierarchical topologies.

Regarding distributed algorithm designs in networks with changing topologies, we believe that the following two problem are practically significant and worth pursuing: fault-tolerant leader election and broadcasting tree maintenance. Leader election problems have been extensively studied for the past ten years. However, there is very little literature addressing this issue under changing-topology environments. The problem is expected to be rather complex as the complexity of a leader election will increase substantially when links or nodes fail.

The broadcasting tree maintenance is about designing an algorithm that can dynamically restructure the current spanning tree used for broadcasting purpose. It will be very beneficial to nodes in the network if a message can be broadcast along the edges of a spanning tree. Good algorithms for such a problem that are able to handle both link failures and node failures are yet to be discovered.

REFERENCES

[1] J. Adams and M. Fishetti. "Star wars SDI: The great experiment," *IEEE Spectrum*, 22(9), pp. 34-64, September 1985.

[2] A. Aho, J. Hopcroft and J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Massachusetts., 1974.

[3] B. Awerbuch and S. Even. "Reliable broadcast protocols in unreliable networks," *Networks*, Vol. 16, pp. 381-396, 1986.

[4] B. Awerbuch. "Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problem," *Proc. of the 19th ACM Symp. on Theory of Computing,* pp. 230-240, May 1987.

[5] A. Baratz, I. Gopal and A. Segall. "Fault tolerant queries in computer networks," *The 2nd International Workshop on Distributed Algorithms.* IEEE, pp. 31-40., July 1987.

[6] R.R. Boorstyn and H. Frank. "Large scale network topological optimization," *IEEE Trans. on Communications*, COM-25(1), pp. 29-47, January 1977.

[7] C. Cheng, I. Cimet and S. Kumar. "A protocol to maintain a minimum spanning tree in a dynamic topology," *ACM SIGCOMM* 88, pp. 330-338, August, 1988.

[8] E.G. Coffman and P.J. Denning. *Operating system Theory*, Prentice-Hall, New Jersey, 1973.

[9] E.G. Coffman. *Computer and Job Shop Scheduling*, John Wiley, New York, 1976.

[10] Y. Dalal. "Broadcast protocols in packet switched computer networks," Ph.D. dissertation, Stanford Univ., Stanford, California, Digital System Lab. Tech. Rep. 128, April 1977.

[11] Y. Dalal and R. Metcalfe. "Reverse path forwarding of broadcast packets," *Commun. Ass. Comput. Mach.*, Vol. 21, pp. 1040-1048, December 1978.

[12] S. Even. "An algorithm for determining whether the connectivity of a graph is at least *k*," *SIAM Journal of Computing*, pp. 393-96, September 1975.

[13] S. Even. *Graph Algorithms*, Computer Science Press, Reading, New York, 1979.

[14] E. Gafni and D. Bertselcas. "Distributed algorithms for generating loop-free routes in networks with frequently changing topologies," *IEEE Trans. on*

Communications, COM-29(1), pp. 11-18, January 1981.

[15] R. Gallager. "An optimal routing algorithm using distributed computation," *IEEE Trans. on Communications*, COM-25, pp. 73-85, 1977.

[16] R. Gallager, P. Humblet, and P. Spira. "A distributed algorithm for minimum-weight spanning trees," *ACM Trans. on Programming Languages and Systems*, Vol. 5, No. 1, January, 1983, pp. 66-77.

[17] J. Garcia-Luna. "A fail-safe routing algorithm for multihop packet-radio networks," *Proc. of IEEE Infocom* 86, pp. 434-443, April 1986.

[18] J. Garcia-Luna-Aceves and R. Rom. "On the dynamic control of network connectivity," *Proc. of IEEE Infocom* 87, pp. 207-217.

[19] M. Gerla and L. Kleinrock. "On the topological design of distributed computer networks," *IEEE Trans. on Communications*, COM-25, January 1977.

[20] L. Kleinrock. *Queueing Systems, Volume 2: Computer Applications*, Wiley-Interscience, New York, 1976.

[21] D. Kleitman. "Methods for investigating the connectivity of large graphs," *IEEE Trans. on Circuit Theory*, CT-16, pp. 232-3, May 1969.

[22] K. Luo, Y. Chow and R. Newman-Wolfe. "An efficient algorithm for reconfiguration of large-scale point-to-point satellite computer networks with maximum connectivity," *The 9th IEEE International Phoenix Conference on Computers and Communications*, pp. 194-201, March 1990.

[23] K. Luo, Y. Chow and R. Newman-Wolfe. "An efficient broadcast protocol in networks with changing topologies," *The 2nd IEEE Workshop on Future Trends of Distributed Computing Systems*, pp. 88-93, September 1990.

[24] K. Luo and Y. Chow. "A message-optimal protocol for global surveys in faulty networks," *IEEE Infocom '91*, pp.525-532, April 1991.

[25] C. McLochlin, C. Ward, Y. C. Chow, R. Newman-Wolfe, J. N. Wilson and T. B. Hughes. "Optimizing the delay and reliability of low altitude satellite network topologies," *IEEE Milcom '87*, October 19-22 1987.

[26] J. McQuillan, I. Richer and E. Rosen. "The new routing algorithm for the APARNET," *IEEE Trans. on Communications*, COM-28, pp. 711-719, 1980.

[27] P. Merlin and A. Segall. "A failsafe distributed routing protocol," *IEEE Trans. on Communications*, COM-27, pp. 1280-1287, 1979.

[28] F. Moss and J. Jaffe. "A responsive distributed routing protocol," *IEEE Trans. on Communications*, COM-30, pp. 1758-1762, 1982.

[29] D. Peleg. "Time-optimal leader election in general networks," *Journal of Parallel and Distributed Computing* 8, pp. 96-99, 1990.

[30] T. Roberttazzi and P. Sarachik. "Self-organizing communication networks," *IEEE Communications Magazine*, Vol. 24, no. 1, pp. 28-33, January 1986.

[31] U. Schumacher. "An algorithm for construction of a k-connected graph with minimum number of edges and quasiminimal diameter," *Networks*, Vol. 14, pp. 63-74, 1984.

[32] A. Segall. "Distributed network protocols," *IEEE Trans. on Inf. Theory*, IT-29, pp. 23-35, 1983.

[33] A. Segall and B. Awerbuch. "A reliable broadcast protocol," *IEEE Trans. on Communications*, COM-31, pp. 895-901, 1983.

[34] A. Tanenbaum. *Computer Networks*. Addison-Wesley, Reading, Massachusetts, 1980.

[35] U. Vishkin. "A distributed orientation algorithm," *IEEE Trans. on Inf. Theory*, Vol. IT-29, pp. 624-629, 1983.

[36] C. Ward. "Topologies and link assignment problems in low altitude satellite networks", Ph.D. dissertation, University of Florida, 1987.

BIOGRAPHICAL SKETCH

Kenneth C.-K. Luo was born in Hsinchu, Taiwan. He received his B.S. degree in computer science and information engineering from National Taiwan University in 1981. He served as a communications officer in the Taiwan Air Force for two years. In 1983, he joined the Institute for Information Industry where he worked as a software engineer. He received his M.S. degree in computer science from the University of California in 1986. He came to the University of Florida in 1987, where he is working on his Ph.D. degree in computer science. His research interest includes distributed computing, computer networks, fault-tolerant computing, combinatorial optimizations and system performance evaluations. He is married to Sze-Mei Ju and has a four-month-old daughter.